Міністерство освіти і науки України
Луцький національний технічний університет

ОСНОВИ ПРОГРАМУВАННЯ МІКРОПРОЦЕСОРІВ ТА ТЕЛЕКОМУНІКАЦІЙНИХ ЗАСОБІВ

Конспект лекцій
для здобувачів першого (бакалаврського) рівня вищої освіти
освітніх програм « Електроніка», «Автомобільна електроніка» та
«Комп'ютеризовані телекомунікаційні мережі»
галузі знань 17 Електроніка, автоматизація та електронні комунікації
спеціальностей 171 Електроніка , та 172 Електронні комунікації та радіотехніка
денної та заочної форм навчання

Луцьк 2025

УДК 004.43(07)

О - 73

Електронна копія друкованого видання передана для внесення в репозитарій ЛНТУ

Директор бібліотеки _____ Наталія ПОЛІЩУК

Рекомендовано до видання вченою радою факультету комп'ютерних та інформаційних технологій ЛНТУ, протокол № __ від « ___» _____2025 року.

Голова вченої ради ФКІТ _____ Інна КОНДІУС

Розглянуто і схвалено на засіданні кафедри електроніки та телекомунікацій ЛНТУ, протокол № ___ від «___» _____2025 року.

Завідувач кафедри ЕіТК	_____	Валентин ЗАБЛОЦЬКИЙ к.т.н., доц. кафедри електроніки та телекомунікацій ЛНТУ
Укладачі:	_____	Микола ХВИЩУН к.ф.-м.н., доц. кафедри електроніки та телекомунікацій ЛНТУ
	_____	Віктор ЛИШУК к.т.н., доц. кафедри електроніки та телекомунікацій ЛНТУ
Рецензент:	_____	Наталія ЛІЩИНА к.т.н., доц., завідувач кафедри інженерії програмного забезпечення ЛНТУ
Відповідальний за випуск:	_____	Валентин ЗАБЛОЦЬКИЙ к.т.н., доц., завідувач кафедри електроніки та телекомунікацій ЛНТУ

О-73 Основи програмування мікропроцесорів та телекомунікаційних засобів. Конспект лекцій для здобувачів першого (бакалаврського) рівня вищої освіти освітніх програм «Електроніка», «Автомобільна електроніка» та «Комп'ютеризовані телекомунікаційні мережі» галузі знань 17 Електроніка, автоматизація та електронні комунікації, спеціальностей 171 Електроніка, та 172 Електронні комунікації та радіотехніка, всіх форм навчання, уклад. М. В. Хвищун, В.В.Лишук. Луцьк: ЛНТУ, 2025.40с.

Видання містить виклад лекційних тем з дисципліни «Основи програмування мікропроцесорів та телекомунікаційних засобів», яке призначене для здобувачів першого (бакалаврського) рівня вищої освіти освітніх програм «Електроніка», «Автомобільна електроніка» та «Комп'ютеризовані телекомунікаційні мережі».

ЗМІСТ

ВСТУП

У сучасному цифровому світі знання програмування стає не просто перевагою, а необхідною складовою професійної підготовки фахівців у багатьох галузях, зокрема електроніки, інформаційних технологій, автоматизації та телекомунікацій. Серед найпоширеніших мов програмування, що використовуються для розробки програмного забезпечення різного рівня – від прикладних додатків до мікроконтролерних систем – ключові позиції займають мови C та Python.

Метою даного конспекту лекцій є ознайомлення студентів з основними принципами програмування мовами C та Python, що включає вивчення базових конструкцій, типів даних, управління потоком виконання, створення функцій, роботи з пам’яттю, обробки файлів, структурування даних та застосування у реальних прикладних завданнях. Особливу увагу приділено порівнянню підходів у цих двох мовах, що дозволить студентам краще зрозуміти їхню специфіку, сильні та слабкі сторони.

Структура курсу передбачає поступове ускладнення матеріалу: від основ алгоритмізації до програмування мікроконтролерів на мові C, включаючи роботу з портами, таймерами, перериваннями, АЦП/ЦАП, обміном даними через UART, SPI та I2C. Усі теми супроводжуються прикладами коду, що дає змогу засвоїти матеріал не лише теоретично, а й на практиці.

Конспект лекцій розрахований на студентів технічних спеціальностей та може бути використаний як для самостійного вивчення, так і як основа для викладання навчального курсу в закладах вищої освіти.

Тема 1. Вступ до програмування мовами C та Python

Програмування – це процес створення інструкцій для комп'ютера, які дозволяють автоматизувати обчислення, аналіз даних, керування пристроями тощо. Воно є основною складовою сучасних інформаційних технологій. Дві з найпопулярніших мов програмування – це мови C та Python, які мають різну історію, синтаксис, і призначення, проте є надзвичайно важливими для інженерів, розробників вбудованих систем, аналітиків даних тощо.

Мова C була розроблена у 1972 році в Bell Labs і відразу завоювала популярність завдяки своїй ефективності та близькості до апаратного рівня. Вона є мовою низького рівня, проте має структури високого рівня, такі як функції, масиви, структури. C дозволяє безпосередньо працювати з пам'яттю та регістрами пристрою, що робить її незамінною у розробці операційних систем, драйверів, вбудованих програм.

Python, у свою чергу, з'явився у 1991 році як мова високого рівня, орієнтована на зручність читання коду та швидкість розробки. Python має синтаксис, близький до природної мови, та є інтерпретованою мовою, що означає можливість негайного виконання коду без компіляції. Це зробило Python надзвичайно популярною у наукових обчисленнях, машинному навчанні, веб-розробці, автоматизації задач.

Порівняння мов C та Python:

– синтаксис: C є суворішим, вимагає оголошення змінних та типів, дужок у структурах керування; Python – простіший та інтуїтивніший;

– типізація: C – статично типізована мова (тип змінної визначається при компіляції); Python – динамічно типізована (тип визначається під час виконання);

– швидкодія: – програми на C зазвичай швидші, бо компілюються у машинний код. Python працює повільніше, але це компенсується зручністю;

– область застосування: C – системне та вбудоване програмування, Python – наука про дані, веб, автоматизація.

Огляд компіляторів та середовищ розробки:

– C: GCC (GNU Compiler Collection), Code::Blocks, Atmel Studio для AVR, Keil для ARM.

– Python: Python IDLE (вбудоване середовище), Visual Studio Code, Jupyter Notebook для аналітики даних.

Перші програми на C та Python: Класичне "Hello, World!" – це програма, яка виводить повідомлення на екран.

Приклад на C:

```
#include <stdio.h>
int main() {
printf("Hello, world!\n");
return 0;
}
```

Приклад на Python:

```
print("Hello, world!")
```

Ці приклади показують простоту синтаксису Python у порівнянні з C, де потрібно оголосити функцію main, підключити бібліотеку stdio.h тощо.

Висновок. Мови C та Python мають свої сильні сторони. Знання обох дозволяє гнучко вирішувати різноманітні завдання – від низькорівневого керування пристроями до аналізу великих масивів даних. У контексті вивчення програмування мікроконтролерів, C має першочергове значення, тоді як Python слугує зручною платформою для симуляцій, аналізу та розробки інтерфейсів.

Список використаних джерел до теми №1

1. Завадський І.А. Програмування мовою C. Харків: Основа, 2024. 278 с.
2. Вишневський А. Основи Python. Київ: Наука і освіта, 2022. 312 с.
3. Kernighan B., Ritchie D. The C Programming Language. Prentice Hall, 2023. 274 p.
4. Matthes E. Python Crash Course. No Starch Press, 2023. 552 p.

Тема 2. Типи даних та змінні в C і Python

У програмуванні типи даних визначають, які значення може приймати змінна і які операції над нею можна виконувати. У мовах програмування C та Python ці підходи суттєво різняться, оскільки C є строго типізованою мовою, а Python – динамічно типізованою. Розуміння особливостей типів даних дозволяє ефективно працювати з пам'яттю, обчисленнями та структурою програм.

Типи даних у мові C: C передбачає використання таких основних типів:

- int – цілі числа (зазвичай 4 байти);
- float – числа з плаваючою комою одинарної точності (4 байти);
- double – числа з плаваючою комою подвійної точності (8 байт);
- char – символи (1 байт);
- bool – логічні значення (включено з stdbool.h з C99).

C також підтримує модифікатори типів, як-от short, long, unsigned, які дозволяють керувати розміром і знаком чисел.

Оголошення змінних обов'язкове перед використанням. Наприклад:

```
int a = 5;
float b = 3.14;
char c = 'A';
```

Типи даних у Python. У Python змінна не має фіксованого типу – тип визначається автоматично на основі присвоєного значення. Основні типи:

- int – цілі числа (довільна довжина);
- float – числа з плаваючою крапкою;
- str – рядки символів;
- bool – логічні значення (True, False).

Наприклад:

```
a = 5          # int
b = 3.14       # float
c = 'A'        # str
d = True       # bool
```

Введення та виведення. У C це реалізується через бібліотеку stdio.h:

```
int x;
printf("Enter a number: ");
scanf("%d", &x);
printf("You entered: %d
", x);
```

У Python все значно простіше:

```
x = int(input("Enter a number: "))
print("You entered:", x)
```

Особливості типізації:

– статична типізація (C): всі змінні повинні мати вказаний тип. Це забезпечує кращу продуктивність та контроль, але вимагає точності;

– динамічна типізація (Python): – дозволяє зручно писати код без попереднього оголошення типів, але потенційно знижує ефективність.

Перетворення типів:

– У C:

```
float x = 10;
int y = (int)x; // явне приведення типу
```

– У Python:

```
x = 10.5
y = int(x)   # перетворення до цілого.
```

Висновок. Вибір типів даних є ключовим при розробці програм, особливо у системному програмуванні (на C), де кожен байт має значення. Python спрощує взаємодію зі змінними, але втрачає в ефективності. Розуміння типів обох мов дає студентам потужний інструмент для оптимальної розробки програмного забезпечення.

Список використаних джерел до теми №2

1. Шевчук В.С. Основи програмування на С. Київ: Ліра-К, 2023. 295 с.
2. Сидоренко Ю.М. Python для інженерів. Львів: Видавництво ЛНУ, 2022. 264 с.
3. Perry G. C Programming Absolute Beginner’s Guide. Pearson, 2023. 368 p.
4. Slatkin B. Effective Python. Addison-Wesley, 2023. 320 p.

Тема 3. Управління потоком виконання в C та Python

Управління потоком виконання – це процес визначення порядку, в якому інструкції програми виконуються. У мовах C і Python існує кілька основних конструкцій керування: умовні оператори (if, else, else if), оператори вибору (switch/case у C, match у Python), а також цикли (for, while, do-while).

Умовні оператори. У мові C умовні оператори мають такий синтаксис:

```
int a = 10;
if (a > 5) {
   printf("a більше за 5\n");
} else if (a == 5) {
   printf("a дорівнює 5\n");
} else {
   printf("a менше за 5\n");
}
```

У мові Python умовні конструкції простіші:

```
a = 10
if a > 5:
   print("a більше за 5")
elif a == 5:
   print("a дорівнює 5")
else:
   print("a менше за 5")
```

Python не використовує дужки або крапки з комами, а структурування виконується через відступи.

Оператор вибору (switch/case): C підтримує оператор switch для вибору між кількома варіантами:

```
int day = 3;
switch (day) {
   case 1:
      printf("Понеділок\n");
```

```
        break;
    case 2:
        printf("Вівторок\n");
        break;
    default:
        printf("Інший день\n");
}
```

У Python з 3.10 з’явився оператор match, який є аналогом switch:

```
day = 3
match day:
    case 1:
        print("Понеділок")
    case 2:
        print("Вівторок")
    case _:
        print("Інший день")
```

Цикли. Циклічне виконання інструкцій дозволяє зменшити повторення коду.

У C:

- for: виконується фіксована кількість разів
- while: виконується поки умова істинна
- do-while: виконується щонайменше один раз

Приклад:

```
for (int i = 0; i < 5; i++) {
    printf("i = %d\n", i);
}
int i = 0;
while (i < 5) {
    printf("i = %d\n", i);
    i++;
}
```

```
int i = 0;
do {
    printf("i = %d\n", i);
    i++;
} while (i < 5);
```

У Python:

```
for i in range(5):
    print("i =", i)
```

```
i = 0
while i < 5:
    print("i =", i)
    i += 1
```

Python не має аналога do-while, але його можна імітувати:

```
i = 0
while True:
    print("i =", i)
    i += 1
    if i >= 5:
        break
```

Оператори управління циклом:

- break – перериває цикл;
- continue – переходить до наступної ітерації.

Висновок. Керування потоком виконання – одна з основ програмування. Хоча синтаксис C і Python відрізняється, логіка структур керування залишається однаковою. Знання цих конструкцій дає змогу створювати складні алгоритми, що реагують на вхідні дані, повторюють обчислення та реалізують розгалуження програмного коду.

Список використаних джерел до теми №3

1. . Білоус В. Основи програмування на Python. Київ: КПІ, 2023. 310 с.
2. Lutz M. Learning Python. O’Reilly Media, 2022. 1648 p.

Тема 4. Функції в C та Python

Функції – це основні будівельні блоки структурованого програмування. Вони дозволяють розбити програму на логічні частини, забезпечити повторне використання коду, зробити програму більш читабельною та зручною для тестування. Як у C, так і в Python функції мають ключову роль у розробці будь-якого застосунку – від простих скриптів до великих систем.

Функції в мові C. Функції оголошуються з визначенням типу значення, яке вони повертають, і типів вхідних параметрів. Наприклад:

```
int add(int a, int b) {
   return a + b;
}
int main() {
   int result = add(3, 4);
   printf("Результат: %d\n", result);
   return 0;
}
```

У C важливо дотримуватися правил оголошення функцій до їх виклику або використовувати прототипи.

Передача параметрів:

- за значенням – копія змінної передається у функцію:

```
void update(int a) {
   a = a + 10;
}
```

- за вказівником – передається адреса змінної:

```
void update(int *a) {
   *a = *a + 10;
}
```

Глобальні та локальні змінні:

- локальні змінні – оголошені всередині функції, доступні тільки в ній;
- глобальні – оголошені поза функціями, доступні у всій програмі.

Рекурсія. Функція викликає сама себе для розв’язання задачі:

```
int factorial(int n) {
   if (n == 0) return 1;
   return n * factorial(n - 1);
}
```

Функції в Python. Python має зручний синтаксис для створення функцій:

```
def add(a, b):
   return a + b
result = add(3, 4)
print("Результат:", result)
```

Типи аргументів у Python:

- позиційні: func(1, 2);
- іменовані (ключові): func(x=1, y=2);
- *args – змінна кількість позиційних аргументів;
- **kwargs – змінна кількість іменованих аргументів.

```
def info(*args, **kwargs):
   for a in args:
      print("ARG:", a)
   for k, v in kwargs.items():
      print(f"{k} = {v}")

info(1, 2, name="Alice", age=25)
```

Глобальні та локальні змінні в Python:

```
x = 10  # глобальна

def test():
   x = 5  # локальна
   print("Локальна x:", x)

print("Глобальна x:", x)
test()
```

Рекурсія в Python:

```
def factorial(n):
    if n == 0:
        return 1
    return n * factorial(n - 1)
```

Висновок. Функції дозволяють створювати модульний код. У C важливо контролювати пам’ять і типи, у Python – зручно працювати з аргументами завдяки *args та **kwargs. Рекурсія – потужний механізм, але вимагає обережності. Знання обох мов дозволяє використовувати сильні сторони кожної для розв’язання широкого спектра задач.

Список використаних джерел до теми №4

1. Григоренко А.О. Програмування мовою С. Дніпро: Академія, 2022. 248 с.
2. Назаренко І.В. Вивчаємо Python. Львів: УАД, 2023. 336 с.
3. Kochan S. Programming in C. Pearson, 2023. 550 p.
4. Hetland M.L. Python Algorithms. Apress, 2021. 300 p.

Тема 5. Масиви, списки та робота з пам’яттю

У процесі розробки програм часто виникає потреба зберігати великі обсяги однотипних даних. У мовах C і Python для цього використовуються різні структури: масиви, списки, кортежі, а також відповідні засоби керування пам’яттю. У C ця тема є критично важливою, адже розробник відповідає за виділення і звільнення пам’яті вручну. У Python же робота з пам’яттю автоматизована, що значно спрощує програмування, але зменшує контроль над ресурсами.

Масиви в C. Масив – це набір елементів одного типу, які зберігаються в суміжних комірках пам’яті.

```
int numbers[5] = {1, 2, 3, 4, 5};
printf("Третій елемент: %d\n", numbers[2]);
```

Масиви можуть бути одно- та багатовимірними:

```
int matrix[2][3] = {{1, 2, 3}, {4, 5, 6}};
```

Вказівники та масиви. У C масиви тісно пов’язані з вказівниками. Ім’я масиву є вказівником на перший елемент:

```
int *ptr = numbers;
printf("Перший елемент: %d\n", *ptr);
```

Динамічне виділення пам’яті. Для створення масивів змінної довжини використовуються функції:

- malloc() – виділення пам’яті;
- free() – звільнення пам’яті.

```
int *arr = (int*)malloc(5 * sizeof(int));
arr[0] = 10;
free(arr);
```

Списки та кортежі в Python. У Python списки – це динамічні структури, які можуть містити елементи різних типів.

```
numbers = [1, 2, 3, 4, 5]
print("Третій елемент:", numbers[2])
```

Списки можна змінювати:

```
numbers.append(6)
```

numbers[0] = 100

Кортежі – це незмінні списки:

tuple_data = (1, 2, 3)

Колекції у Python:

- list – список
- tuple – кортеж
- set – множина
- dict – словник (ключ-значення)

student = {"name": "Ivan", "age": 21}

print(student["name"])

Управління пам’яттю у Python. Python автоматично керує пам’яттю за допомогою збирача сміття (garbage collector). Це звільняє програміста від необхідності явно звільняти пам’ять, але у великих проєктах важливо враховувати ефективність використання пам’яті.

Висновок. Робота з масивами та пам’яттю у C потребує точності й уважності – помилки в управлінні пам’яттю призводять до збоїв. Python забезпечує зручний та безпечний спосіб роботи з колекціями даних, зменшуючи складність, але й контроль. Знання обох підходів дозволяє майбутнім розробникам ефективно працювати як з системними ресурсами, так і з прикладними задачами.

Список використаних джерел до теми №5

1. Федоренко І.О. Масиви та вказівники в мові С. Київ: ВНТУ, 2021. 216 с.
2. Савчук М.В. Python: Робота з колекціями. Львів: НУ ЛП, 2022. 240 с.
3. Zelle J. Python Programming: An Introduction to Computer Science. Franklin, Beedle & Associates, 2023. 520 p.
4. Oualline S. Practical C Programming. O'Reilly Media, 2021. 456 p.

Тема 6. Робота з файлами в C та Python

У програмуванні робота з файлами є одним із ключових елементів для збереження, обміну та обробки даних. Мови C та Python мають вбудовані засоби для взаємодії з файлами, що дозволяє зчитувати та записувати інформацію у текстовому або бінарному форматі. Розуміння цих механізмів необхідне для створення надійного ПЗ, яке працює з конфігураціями, журналами подій, базами даних тощо.

Файли в C. Мова C надає низькорівневий доступ до файлової системи через бібліотеку stdio.h:

- fopen() – відкриває файл;
- fprintf() / fscanf() – робота з текстовими файлами;
- fread() / fwrite() – для бінарних файлів;
- fclose() – закриває файл.

Робота з текстовим файлом (C):

```
FILE *fp = fopen("data.txt", "w");
if (fp != NULL) {
   fprintf(fp, "Привіт, файл!\n");
   fclose(fp);
}
```

Читання з файлу:

```
char buffer[100];
FILE *fp = fopen("data.txt", "r");
if (fp != NULL) {
   fgets(buffer, 100, fp);
   printf("Прочитано: %s", buffer);
   fclose(fp);
}
```

Бінарні файли в C:

```
struct Person {
   char name[20];
```

```
    int age;
};

struct Person p = {"Ivan", 30};
FILE *fp = fopen("person.dat", "wb");
fwrite(&p, sizeof(struct Person), 1, fp);
fclose(fp);
```

Файли в Python. Python використовує вбудовану функцію open():

```
with open("data.txt", "w") as file:
    file.write("Привіт, файл!\n")
```

Читання з текстового файлу:

```
with open("data.txt", "r") as file:
    content = file.read()
    print("Прочитано:", content)
```

Запис у бінарний файл:

```
with open("data.bin", "wb") as file:
    file.write(bytes([100, 200, 50]))
```

Серіалізація даних у Python:

- pickle – зберігає об’єкти Python у файл у бінарному вигляді;
- json – для обміну даними між додатками.

Приклад з pickle:

```
import pickle

data = {"name": "Ivan", "age": 30}
with open("data.pkl", "wb") as f:
    pickle.dump(data, f)
```

Читання з JSON:

```
import json

with open("data.json", "r") as f:
    data = json.load(f)
```

```
print(data["name"])
```

Обробка помилок:

- у C перевіряють вказівник FILE * на NULL;
- у Python використовують конструкцію try...except.

Висновок. Робота з файлами забезпечує збереження інформації між сесіями програми. C дає змогу точно керувати файлами, у тому числі на рівні байтів. Python надає вищий рівень абстракції та сучасні засоби серіалізації. Навички роботи з файлами є обов'язковими для будь-якого програміста.

Список використаних джерел до теми №6

1. Ткачук А.П. Робота з файлами в C. Тернопіль: Едельвейс, 2021. 214 с.
2. Іванченко С.М. Python: введення у файлові структури. Київ: КНУ, 2023. 192 с.
3. Summerfield M. Programming in Python 3. Addison-Wesley, 2023. 648 p.
4. Kochan S. Programming in C. Pearson, 2023. 550 p.

Тема 7. Робота зі структурами даних та об'єктно-орієнтоване програмування (ООП) у Python

У мові C та Python існують різні способи організації складних типів даних. У C це структури та об'єднання, які дозволяють групувати дані різних типів. У Python реалізовано повноцінну підтримку об'єктно-орієнтованого програмування (ООП), що дозволяє створювати класи, наслідування, інкапсуляцію, поліморфізм.
Структури в мові C: Структура (struct) – це агрегований тип даних, який дозволяє об'єднати змінні різних типів:

```
struct Student {
   char name[50];
   int age;
   float grade;
};

struct Student s1 = {"Ivan", 20, 89.5};
printf("Ім'я: %s, Вік: %d\n", s1.name, s1.age);
```

Використання структур у мікроконтролерах. Структури широко застосовуються для зберігання даних датчиків, регістрів тощо:

```
struct SensorData {
   uint16_t temperature;
   uint16_t humidity;
};
```

Об'єднання (union):

```
union Data {
   int i;
   float f;
};
```

У union всі поля розміщуються в одній області пам’яті – це корисно для економії ресурсів, але вимагає обережності.

ООП у Python. Об’єктно-орієнтоване програмування базується на створенні класів та об’єктів:

```
class Student:
   def __init__(self, name, age, grade):
      self.name = name
      self.age = age
      self.grade = grade

   def display(self):
      print(f"Ім’я: {self.name}, Вік: {self.age}, Оцінка: {self.grade}")

s1 = Student("Ivan", 20, 89.5)
s1.display()
```

Наслідування:

```
class Person:
   def __init__(self, name):
      self.name = name
```

```
class Student(Person):
    def study(self):
        print(f"{self.name} вчиться")


s = Student("Olena")
s.study()
```

Інкапсуляція і поліморфізм:

- інкапсуляція – приховування внутрішньої реалізації (через _ та __);
- поліморфізм – використання одного інтерфейсу для різних типів об’єктів.

Порівняння структур у С та класів у Python:

Ознака	С (struct)	Python (class)
Поля/атрибути	є	є
Методи	немає	є
Наслідування	немає	є
Інкапсуляція	обмежена	підтримується
Поліморфізм	немає	є

Висновок. С дозволяє ефективно працювати зі структурами даних на рівні пам’яті, особливо у вбудованих системах. Python реалізує повноцінну ООП-модель, зручно адаптовану для розробки складних застосунків. Розуміння обох підходів розширює можливості розробника.

Список використаних джерел до теми №7

1. Савчук В.О. Структури в С. Київ: Ліра-К, 2022. 198 с.
2. Мельник Р.В. Об’єктно-орієнтоване програмування на Python. Харків: Фактор, 2023. 272 с.
3. Hetland M.L. Beginning Python. Apress, 2023. 648 p.
4. King K.N. C Programming: A Modern Approach. W. W. Norton & Company, 2022. 832 p.

Тема 8. Робота з бітовими операціями в С

Бітові операції – це операції, які безпосередньо маніпулюють окремими бітами чисел. Вони є основою низькорівневого програмування, особливо у мікроконтролерах, де часто потрібно працювати з регістрами вводу/виводу, керувати окремими бітами стану, перевіряти або змінювати прапори. Мова С надає повний набір побітових операторів, що дозволяє ефективно працювати з апаратним забезпеченням.

Основні бітові оператори в С:

- & – AND (побітове І);
- | – OR (побітове АБО);
- ^ – XOR (виключне АБО);
- ~ – NOT (заперечення);
- << – зсув вліво;
- >> – зсув вправо.

Приклади використання:

```
unsigned char a = 0b1100;
unsigned char b = 0b1010;
unsigned char result;

result = a & b;  // 0b1000
result = a | b;  // 0b1110
result = a ^ b;  // 0b0110
result = ~a;     // 0b0011 (у 8-бітному представленні)
result = a << 1; // 0b10000
result = b >> 1; // 0b0101
```

Бітові маски. Бітові маски дозволяють керувати окремими бітами змінної. Наприклад, щоб встановити, скинути або перевірити конкретний біт:

```
#define BIT3 (1 << 3)

unsigned char port = 0x00;
```

```
port |= BIT3;    // встановити 3-й біт
port &= ~BIT3;   // скинути 3-й біт
if (port & BIT3) // перевірити 3-й біт
{
   // виконати дію
}
```

Робота з регістрами мікроконтролерів. Налаштування портів вводу/виводу в мікроконтролерах відбувається через пряме встановлення або скидання бітів у спеціальних регістрах:

```
DDRB |= (1 << PB0);  // встановити PB0 як вихід
PORTB &= ~(1 << PB0); // скинути рівень на PB0 (лог. 0)
PORTB |= (1 << PB0);  // встановити рівень на PB0 (лог. 1)
```

Практичне застосування:

- керування станами світлодіодів;
- зчитування статусу кнопок;
- побудова простих протоколів зв’язку.
- оптимізація пам’яті через зберігання кількох прапорів у одному байті.

Висновок. Бітові операції забезпечують ефективний та гнучкий контроль на рівні окремих бітів. Це необхідний інструмент для розробки програмного забезпечення мікроконтролерів та інших вбудованих систем. Розуміння принципів побітової логіки дозволяє оптимізувати як продуктивність, так і обсяг пам’яті, що використовується.

Список використаних джерел до теми №8

1. Твердохліб Ю. Побітові операції та керування портами в С. Вінниця: НУВГП, 2021. 124 с.

2. Дьяків С.М. Основи програмування мікроконтролерів на С. Львів: ЛНУ, 2022. 148 с.

3. Pont M. Embedded C. Pearson Education, 2023. 672 p.

4. Barrett S.F., Pack D.J. Embedded Systems Design with C. Springer, 2021. 278 p.

Тема 9. Робота з перериваннями в C (AVR, Arduino)

Переривання (interrupts) – це механізм, який дозволяє мікроконтролеру тимчасово призупинити виконання основної програми, щоб обробити подію, що надійшла ззовні або від внутрішніх модулів. Після обробки переривання управління повертається до основної програми. Це дозволяє ефективно реагувати на події в режимі реального часу.

У мікроконтролерах AVR та платформах Arduino переривання широко застосовуються для обробки сигналів від кнопок, давачів, таймерів, UART тощо.

Види переривань:

1. Апаратні – генеруються зовнішніми або внутрішніми пристроями (таймери, UART, кнопки).
2. Програмні – викликаються штучно в коді (рідко застосовуються в мікроконтролерах).

Робота з перериваннями в C для AVR. Для роботи з перериваннями використовується бібліотека <avr/interrupt.h>, яка містить макрос ISR() для створення обробників переривань.

1. Підключення бібліотек і дозволи:

```
#include <avr/io.h>
#include <avr/interrupt.h>

sei(); // Дозвіл глобальних переривань
cli(); // Заборона глобальних переривань
```

2. Обробник переривання:

```
ISR(INT0_vect) {
   // Дії при надходженні переривання від INT0
}
```

3. Налаштування INT0:

```
GICR |= (1 << INT0);     // Дозволити INT0
MCUCR |= (1 << ISC01);   // Спадний фронт сигналу
```

Робота з таймерами і переповненням:

```
TCCR0 |= (1 << CS02);     // Таймер з передділенням
TIMSK |= (1 << TOIE0);    // Дозвіл переривання від переповнення

ISR(TIMER0_OVF_vect) {
   // Дії при переповненні таймера
}
```

Робота з перериваннями в Arduino. Arduino IDE спрощує роботу з перериваннями за допомогою функції attachInterrupt():

```
void setup() {
  pinMode(13, OUTPUT);
  attachInterrupt(digitalPinToInterrupt(2), blink, FALLING);
}

void blink() {
  digitalWrite(13, !digitalRead(13));
}
```

Ключові особливості використання ISR:

- ISR не повинна містити довготривалих операцій;
- Заборонено використовувати delay() усередині ISR;
- Змінні, які використовуються як у ISR, так і в основному коді, мають бути volatile.

Приклади практичного використання:

- миттєве реагування на натискання кнопки;
- створення програмних лічильників на базі таймера;
- асинхронне зчитування UART-даних;
- формування сигналів ШІМ із заданим інтервалом.

Висновок. Переривання забезпечують швидку та ефективну реакцію на події в системах реального часу. Вивчення принципів їх роботи є важливим для розробників вбудованих систем, оскільки дозволяє оптимізувати продуктивність і точність виконання програм.

Список використаних джерел до теми №9

1. Савчук І. Програмування AVR: таймери та переривання. Тернопіль: ФОП Савчук, 2021. 196 с.
2. Павлюк М.В. Основи Arduino та робота з подіями. Львів: ЛНУ, 2022. 184 с.
3. Barnett R.H. Embedded C Programming and the Atmel AVR. Delmar Cengage Learning, 2023. 560 p.
4. Dhananjay V.G. Advanced Microcontrollers. McGraw-Hill Education, 2021. 468 p.

Тема 10. Робота з модулями та бібліотеками в C та Python

Для створення масштабованих програм важливо розділяти код на частини, які можна використовувати повторно – модулі й бібліотеки. Це дозволяє організовувати програму логічно, спрощувати супровід і покращувати читаність. У мові C це здійснюється через заголовочні файли (.h) та реалізаційні файли (.c), а в Python – через модулі (.py) та пакети.

Модульність у C. Код поділяється на:

- .h – заголовочний файл з оголошеннями функцій, структур, макросів;
- .c – файл з реалізацією функцій.

Приклад створення бібліотеки (C): Файл led.h:

```
#ifndef LED_H
#define LED_H

void led_on(void);
void led_off(void);

#endif
```

Файл led.c:

```
#include <avr/io.h>
#include "led.h"

void led_on() {
   PORTB |= (1 << PB0);
```

```
}

void led_off() {
   PORTB &= ~(1 << PB0);
}
```

Підключення в основному коді:

```
#include "led.h"

int main() {
   led_on();
   // ...
}
```

Модулі та пакети в Python. Кожен .py файл – модуль. Модулі можна імпортувати за допомогою import, from ... import, або as:

```
import math
print(math.sqrt(16))
```

Створення власного модуля: Файл mymodule.py:

```
def greet(name):
   print(f"Привіт, {name}!")
```

У основному коді:

```
import mymodule
mymodule.greet("Іван")
```

Головна функція Python:

```
if __name__ == "__main__":
   print("Це виконується при запуску напряму")
```

Написання бібліотеки для сенсора (приклад):

```
# dht11.py
import time

def read_sensor():
   return {"temp": 23.5, "humidity": 60}
```

```
# main.py
import dht11
result = dht11.read_sensor()
print(result)
```

Висновок. Розбиття коду на модулі й бібліотеки сприяє структурованості, повторному використанню коду та зручному тестуванню. У C це дозволяє створювати окремі драйвери та логічні блоки, а в Python – організовувати логіку в окремі файли та каталоги.

Список використаних джерел до теми №10

1. Лисенко В.О. Бібліотеки та модулі в C. Київ: КПІ, 2023. 204 с.
2. Матвієнко О.М. Python: створення модулів. Харків: ХНУРЕ, 2022. 220 с.
3. Lutz M. Python Pocket Reference. O'Reilly, 2023. 296 p.
4. Kochan S. Programming in C. Pearson, 2023. 550 p.

Тема 11. Оптимізація коду та налагодження

Оптимізація коду та процес його налагодження є важливими етапами розробки надійного та ефективного програмного забезпечення. Оптимізація дозволяє зменшити споживання пам’яті, підвищити швидкість виконання програм, а налагодження – виявити й усунути помилки. І в C, і в Python ці процеси мають свої особливості.

Оптимізація в мові C. Оптимізація на рівні C-програмування передбачає:

- використання статичних змінних для збереження стану;
- уникнення зайвих викликів функцій у циклах;
- застосування побітових операцій замість арифметичних (де це доцільно);
- виключення зайвого копіювання даних.

Приклад оптимізації:

```
// Неоптимізовано:
for (int i = 0; i < strlen(str); i++) {
    ...
}
```

```
// Оптимізовано:
int len = strlen(str);
for (int i = 0; i < len; i++) {
   ...
}
```

Оптимізація для вбудованих систем:

- зменшення розміру коду (відключення непотрібних бібліотек);
- економія енергії (режими сну);
- використання const для змінних у Flash.

Профілювання програм (C). Профілювальники дозволяють визначити, які частини коду споживають найбільше часу:

- gprof – GNU Profiler;
- valgrind – перевірка пам’яті та витоків.

Налагодження в C:

- GDB – GNU Debugger дозволяє покроково виконувати програму, переглядати значення змінних;
- printf() – часто використовується для виводу діагностичної інформації;
- LED-індикація у вбудованих системах – простий спосіб сигналізації.

Налагодження в Python:

- Вбудований модуль pdb для покрокового проходження коду;
- IDE (PyCharm, VS Code) з інтерактивним дебагером;
- Модуль logging для збереження журналів подій.

Використання logging:

```
import logging
logging.basicConfig(level=logging.INFO)
logging.info("Це повідомлення для налагодження")
```

Обробка винятків у Python:

```
try:
   x = 1 / 0
except ZeroDivisionError as e:
   print("Помилка:", e)
```

Зменшення споживання ресурсів:

– використання генераторів замість списків:

```
# Генератор
for x in (i for i in range(1000000)):
  pass
```

– мемоізація обчислень (через декоратори @lru_cache).

Висновок. Оптимізація коду та налагодження – це процеси, що роблять програму ефективною, стабільною та зручною в обслуговуванні. У мовах C та Python є спеціалізовані засоби для виявлення вузьких місць, тестування логіки, контролю витоків пам’яті та документування помилок.

Список використаних джерел до теми №11

1. Гнатюк М. Оптимізація програмного коду. Київ: КПІ, 2021. 192 с.
2. Шевченко Р. Налагодження програм у мікроконтролерах. Львів: ЛНТУ, 2022. 164 с.
3. Martelli A. Python in a Nutshell. O'Reilly, 2023. 772 p.
4. Stallman R. Debugging with GDB. Free Software Foundation, 2022. 420 p.

Тема 12. Програмування мікропроцесорів на асемблері

Асемблер – це мова низького рівня, яка відображає машинні інструкції процесора в зручному для людини форматі. Вона дозволяє безпосередньо керувати регістрами, пам’яттю та портами вводу/виводу мікроконтролера. Програмування на асемблері вимагає знань про архітектуру мікропроцесора, проте забезпечує максимальну ефективність і контроль над ресурсами системи.

Архітектура процесора.Типовий мікропроцесор складається з:

– арифметично-логічного пристрою (ALU) – виконує арифметичні та логічні операції;

– регістрів загального призначення (R0–Rn) – для збереження тимчасових значень;

– лічильника команд (PC) – вказує на адресу наступної інструкції;

– стеку (SP) – зберігає адреси під час викликів функцій.

Синтаксис асемблера (на прикладі AVR):

```
LDI R16, 0x05      ; завантажити в регістр R16 число 5
LDI R17, 0x03      ; завантажити в R17 число 3
ADD R16, R17       ; скласти R16 + R17
OUT PORTB, R16     ; вивести результат у порт B
```

Типові інструкції асемблера:

- LDI – завантаження константи в регістр;
- MOV – копіювання між регістрами;
- ADD, SUB – арифметичні операції;
- AND, OR, EOR – логічні операції;
- IN, OUT – доступ до периферійних регістрів;
- RJMP, CALL, RET – переходи та виклики.

Робота з пам’яттю:

- AVR має поділ пам’яті на Flash (програма), SRAM (дані), EEPROM (постійні дані);
- Асемблер дозволяє керувати пам’яттю вручну:

```
LDI R30, LOW(RAMEND)
LDI R31, HIGH(RAMEND)
```

Приклад програми – блимаючий світлодіод на PB0

```
.ORG 0x00
RJMP INIT

INIT:
   LDI R16, 0x01
   OUT DDRB, R16     ; PB0 як вихід
LOOP:
   OUT PORTB, R16
   CALL DELAY
   EOR R16, 0x01
   RJMP LOOP
```

```
DELAY:
   LDI R18, 0xFF
   LDI R19, 0xFF
D1:
   DEC R19
   BRNE D1
   DEC R18
   BRNE D1
   RET
```

Переваги програмування на асемблері:

- максимальна швидкість та контроль над часом виконання;
- мінімальний обсяг пам’яті;
- точна робота з периферією.

Недоліки:

- важче у читанні та супроводі;
- більше часу на розробку;
- потребує глибоких знань архітектури.

Застосування:

- критичні частини коду (наприклад, ініціалізація або обробка переривань);
- написання драйверів;
- завантажувачі (bootloaders).

Висновок. Асемблерне програмування є важливою складовою вбудованих систем. Хоча воно потребує більш глибоких знань, воно відкриває можливість повного контролю над апаратною частиною. Розуміння асемблера також допомагає краще писати та оптимізувати програми на мові С.

Список використаних джерел до теми №12

1. Климчук Ю.М. Мікропроцесори і асемблер. Львів: ЛНУ, 2023. 232 с.
2. Mazidi M.A., Causey R.D. AVR Microcontroller and Embedded Systems. Pearson, 2023. 688 p.
3. Barnett R.H. Introduction to Embedded Systems. Delmar Cengage Learning, 2022. 640 p.

Тема 13. Архітектура мікроконтролерів Atmega та середовище розробки. Робота з портами введення/виведення (GPIO) у С

Мікроконтролери Atmega (наприклад, Atmega328P, Atmega32) належать до лінійки AVR від компанії Atmel (тепер Microchip Technology). Вони мають гарну підтримку з боку спільноти, прості в освоєнні та широко використовуються в навчанні, хобі-розробках і промислових системах.

Основні компоненти архітектури Atmega:

- ALU (арифметико-логічний пристрій);
- Flash-пам’ять – для зберігання програми;
- SRAM – оперативна пам’ять для змінних;
- EEPROM – постійна пам’ять для збереження даних;
- I/O-порти (GPIO) – для зчитування/виведення сигналів;
- Таймери, UART, SPI, I2C, АЦП, ШІМ – вбудовані периферійні модулі.

Середовища розробки для Atmega:

1. Atmel Studio – офіційне середовище від Microchip (підтримка C/ASM);
2. AVR-GCC + AVRDude – компілятор та інструмент для завантаження прошивки;
3. Arduino IDE – зручна оболонка для мікроконтролерів, зокрема Atmega328P;
4. Proteus – для моделювання роботи мікроконтролерів.

Робота з портами GPIO в C. Порти вводу/виводу дозволяють керувати зовнішніми пристроями – кнопками, світлодіодами, реле тощо.

Налаштування порту як виходу:

```
DDRB |= (1 << PB0); // встановити PB0 як вихід
```

Увімкнення/вимкнення логічного рівня:

```
PORTB |= (1 << PB0);  // подати логічну 1 на PB0
PORTB &= ~(1 << PB0); // скинути PB0 в 0
```

Читання з входу:

```
DDRD &= ~(1 << PD2); // PD2 як вхід
if (PIND & (1 << PD2)) {
```

```
    // натиснута кнопка
}
```

Приклад: вмикання світлодіода при натисканні кнопки

```
#define LED_PIN PB0
#define BTN_PIN PD2

int main(void) {
    DDRB |= (1 << LED_PIN);
    DDRD &= ~(1 << BTN_PIN);

    while (1) {
        if (PIND & (1 << BTN_PIN))
            PORTB |= (1 << LED_PIN);
        else
            PORTB &= ~(1 << LED_PIN);
    }
}
```

Bitwise-операції для керування GPIO:

- |= – встановлення біта;
- &=~ – очищення біта;
- ^= – інверсія біта;
- & – перевірка стану біта.

Особливості GPIO в Atmega:

- доступні внутрішні підтягувальні резистори;
- можна конфігурувати напрямок і стан порту;
- GPIO працюють швидко і стабільно без зовнішніх драйверів.

Висновок. Мікроконтролери Atmega забезпечують чудовий баланс між функціональністю та простотою використання. Робота з GPIO дозволяє реалізовувати безліч проектів: від простих індикаторів до складних автоматизованих систем. Середовище Atmel Studio дає змогу писати ефективні програми мовою C з повним доступом до апаратної частини.

Список використаних джерел до теми №13

1. Сидоренко А.О. Програмування мікроконтролерів AVR. Київ: КПІ, 2022. 216 с.
2. Тарасенко Ю.В. Робота з GPIO в Atmega. Львів: Видавництво ЛНУ, 2023. 208 с.
3. Barnett R.H. Embedded C Programming and the Atmel AVR. Cengage Learning, 2023. 560 p.
4. Dean J.W. AVR Microcontroller and Embedded Systems. Pearson, 2022. 736 p.

Тема 14. Таймери та переривання в мікроконтролерах AVR

Таймери є невід'ємною частиною мікроконтролерів. Вони дозволяють вимірювати час, генерувати періодичні події, створювати затримки, генерувати сигнали ШІМ. У поєднанні з перериваннями таймери забезпечують роботу програм у режимі реального часу без затримок основного циклу.

Мікроконтролери AVR мають кілька таймерів: 8-бітні (Timer0, Timer2) і 16-бітні (Timer1). Кожен з них має свій набір регістрів керування.

Регістри таймера (на прикладі Timer0):

- TCCR0 – регістр керування;
- TCNT0 – лічильник таймера;
- OCR0 – регістр порівняння;
- TIMSK – маска дозволу переривань;
- TIFR – прапори переривань.

Режими роботи таймера:

1. Normal Mode – лічильник зростає до 255, потім переповнюється.
2. CTC Mode (Clear Timer on Compare) – лічильник скидається при збігу з OCRx.
3. PWM (Fast PWM, Phase Correct PWM) – генерація імпульсів змінної ширини.

Налаштування таймера в режимі Normal і CTC:

```
TCCR0 |= (1 << CS01) | (1 << CS00); // передділення 64
TIMSK |= (1 << TOIE0);              // дозвіл переповнення
ISR(TIMER0_OVF_vect) {
```

```
   // обробка переповнення
}
TCCR0 |= (1 << WGM01);            // режим CTC
OCR0 = 124;                       // значення порівняння
TIMSK |= (1 << OCIE0);            // дозвіл переривання по збігу
ISR(TIMER0_COMP_vect) {
   // обробка порівняння
}
```

PWM-сигнал на Timer0:

```
DDRB |= (1 << PB3);
TCCR0 |= (1 << WGM00) | (1 << WGM01) | (1 << COM01) | (1 << CS01);
OCR0 = 128; // 50% ШІМ
```

Переваги використання таймерів з перериваннями:

- точне керування подіями у часі;
- зменшення навантаження на основний цикл програми;
- реалізація безперервного обліку часу.

Рекомендації:

- використовуйте volatile для змінних, які змінюються в ISR;
- не використовуйте delay() у ISR;
- уникайте довгих операцій у перериваннях – вони блокують інші ISR.

Приклад: періодичне блимання світлодіода через таймер

```
#define LED_PIN PB0
ISR(TIMER0_OVF_vect) {
   PORTB ^= (1 << LED_PIN);
}

int main(void) {
   DDRB |= (1 << LED_PIN);
   TCCR0 |= (1 << CS02); // передділення 256
   TIMSK |= (1 << TOIE0);
   sei();
```

```
  while (1) {
    // інші дії
  }
}
```

Висновок. Таймери і переривання є потужними інструментами для реалізації задач реального часу. У мікроконтролерах AVR вони дозволяють виконувати точне вимірювання інтервалів, реалізувати асинхронну логіку та генерувати сигнали. Їх ефективне використання покращує продуктивність і гнучкість програмованої системи.

Список використаних джерел до теми №14

1. Довбня С.М. Таймери і переривання в AVR. Київ: Основа, 2021. 172 с.
2. Павлюк І.І. Розробка систем на AVR. Львів: ЛНТУ, 2023. 198 с.
3. Barnett R.H. Embedded C Programming and the Atmel AVR. Cengage Learning, 2023. 560 p.
4. Dean J.W. AVR Microcontroller and Embedded Systems. Pearson, 2022. 736 p.

Тема 15. Робота з АЦП та ЦАП у С. Організація обміну даними через UART, I2C, SPI

Мікроконтролери мають вбудовані периферійні модулі, які дозволяють працювати з аналоговими сигналами та здійснювати обмін даними з іншими пристроями. Найпоширенішими серед них є АЦП (аналогово-цифровий перетворювач), ЦАП (цифро-аналоговий перетворювач), а також інтерфейси UART, I2C та SPI.

1. Аналогово-цифровий перетворювач (АЦП). АЦП дозволяє зчитувати аналогові сигнали з сенсорів (наприклад, температури, світла) і перетворювати їх у цифрове значення для подальшої обробки мікроконтролером.

Налаштування АЦП в AVR (наприклад, Atmega328P):

```
ADMUX = (1 << REFS0); // опорна напруга – AVCC
ADCSRA = (1 << ADEN) | (1 << ADSC) | (1 << ADPS2); // увімкнути АЦП, запуск конверсії
```

```
while (ADCSRA & (1 << ADSC)); // чекати завершення
uint16_t value = ADC;
```

2. Цифро-аналоговий перетворювач (ЦАП). Більшість AVR не мають вбудованого ЦАП, проте можна використовувати:

- ШІМ (PWM) із фільтром нижніх частот для генерації аналогового сигналу;
- зовнішній ЦАП (наприклад, MCP4725) через I2C.

Приклад генерації аналогового сигналу через PWM:

```
DDRD |= (1 << PD6);
TCCR0A |= (1 << COM0A1) | (1 << WGM00) | (1 << WGM01);
TCCR0B |= (1 << CS01);
OCR0A = 128; // 50% ШІМ (аналог ~2.5 В)
```

3. UART (Universal Asynchronous Receiver/Transmitter): UART – простий послідовний інтерфейс обміну даними.

Налаштування UART:

```
UBRR0 = 103; // для 9600 бод @ 16 МГц
UCSR0B = (1 << TXEN0) | (1 << RXEN0);
UCSR0C = (1 << UCSZ01) | (1 << UCSZ00);

// передача символу
while (!(UCSR0A & (1 << UDRE0)));
UDR0 = 'A';

// прийом символу
while (!(UCSR0A & (1 << RXC0)));
char c = UDR0;
```

4. I2C (TWI - Two Wire Interface): I2C – дволінійний протокол обміну, часто використовується з сенсорами або зовнішніми ЦАП.

Налаштування I2C (у C):

```
TWBR = 32; // частота ~100 кГц
// старт, адресація, передача/прийом – через TWI регістри
```

У реальних програмах рекомендується використовувати бібліотеки (наприклад, i2cmaster.h).

5. SPI (Serial Peripheral Interface): SPI – високошвидкісний синхронний інтерфейс для обміну даними з модулями (наприклад, SD-карти, дисплеї).

Налаштування SPI (майстер):

```
DDRB |= (1 << PB5) | (1 << PB7) | (1 << PB4); // SCK, MOSI, SS
SPCR = (1 << SPE) | (1 << MSTR) | (1 << SPR0);

// передача байта
SPDR = 0x55;
while (!(SPSR & (1 << SPIF)));
```

Переваги:

- UART – просто і зручно, ідеально для терміналів;
- I2C – багатопристроєвий, мале навантаження на лінії;
- SPI – дуже швидкий і надійний.

Висновок: Використання АЦП та цифрових інтерфейсів UART, I2C, SPI є основою комунікації вбудованих систем. Вони дозволяють зчитувати дані з сенсорів, передавати їх у зовнішні системи або керувати іншими пристроями. Їх знання необхідне для побудови повноцінних інтегрованих рішень.

Список використаних джерел до теми №15

1. Коваленко С.А. Інтерфейси UART, SPI, I2C у мікроконтролерах. Київ: Ліра-К, 2023. 204 с.
2. Шевченко І.М. АЦП і ЦАП у вбудованих системах. Харків: ХНУРЕ, 2022. 192 с.
3. Gay W., Practical AVR Microcontrollers. Apress, 2022. 408 p.
4. Morton J. AVR: An Introductory Course. Newnes, 2021. 320 p.

О -73

Основи програмування мікропроцесорів та телекомунікаційних засобів. Конспект лекцій для здобувачів першого (бакалаврського) рівня вищої освіти освітніх програм «Електроніка», «Автомобільна електроніка» та «Комп'ютеризовані телекомунікаційні мережі» галузі знань 17 Електроніка, автоматизація та електронні комунікації, спеціальностей 171 Електроніка та 172 Електронні комунікації та радіотехніка, всіх форм навчання, уклад. М. В. Хвищун, В.В.Литшук. Луцьк: ЛНТУ, 2025. 40 с.

Комп'ютерний набір Микола ХВИЩУН
Редактор Віктор ЛИШУК

Підп. до друку «__»_____2025 р.
Формат 60х84/16. Папір офс.
Гарн. Таймс. Ум. друк. арк. 1,67.
Тираж 50 прим.

Відділ іміджу та промоції
Луцького національного технічного університету
43018 м. Луцьк, вул. Львівська, 75
Друк – ВІП ЛНТУ