НАЦІОНАЛЬНИЙ АВІАЦІЙНИЙ УНІВЕРСИТЕТ

Кафедра електроніки, робототехніки, технологій моніторингу

та Інтернету речей

Модульна контрольна робота-1 з дисципліни

**КОДУВАННЯ І ДЕКОДУВАННЯ СИГНАЛІВ В ЕЛЕКТРОННИХ ТЕХНОЛОГІЯХ**

Виконав студент групи ЗБ-234Бстн

Кушніренко Юрій

«20»\_\_\_\_травня\_\_\_\_2023

Перевірив проф. Білецький А.Я.

«\_\_\_\_»\_\_\_\_\_\_\_\_\_\_2023

Київ – 2023

ЗМІСТ

[1. ЗАВДАННЯ: 1](#_Toc138078343)

[2. ХІД РОБОТИ 2](#_Toc138078344)

[2.1. Алгоритм роботи програми можна описати так: 3](#_Toc138078345)

[2.1.1 Імпортуємо необхідні модулі та бібліотеки: 3](#_Toc138078346)

[2.1.2 Визначимо глобальні змінні[3]: 3](#_Toc138078347)

[2.1.3 Визначимо клас HistoryWindow[2,3]: 3](#_Toc138078348)

[2.1.4 Визначимо клас ErrorWindow[2,3]: 3](#_Toc138078349)

[2.1.5 Визначимо основну точку входу програми[2]: 4](#_Toc138078350)

[3. ВИСНОВКИ 5](#_Toc138078351)

[4. ДОДАТОК (Лістинг програми) 6](#_Toc138078352)

[5. ЛІТЕРАТУРА 15](#_Toc138078353)

# ЗАВДАННЯ:

Алгоритмічне і програмне забезпечення генерації псевдовипадкових послідовностей на основі узагальнених матриць Галуа.

# ХІД РОБОТИ

Для вирішення задачі синтезу матриць ми застосовуємо узагальнене правило діагонального заповнення. Суть цього правила полягає в наступному: ми починаємо з нижнього рядка матриці G і записуємо в неї примітивний(утворюючий) елемент поля ω, який є елементом поля GF(2^n) над вибраним поліномом fn. Елементи рядка, розташовані ліворуч від ω, заповнюються нулями. Потім наступні рядки матриці формуються шляхом зсуву попереднього рядка на один розряд вліво. Якщо при цьому старший ненульовий розряд рядка виходить за межі матриці, вектори, відповідні таким рядкам, приводяться до залишку по модулю fn, що робить рядок матриці n-розрядним.[1]

На рисунку представлено структурну схему узагальненого базового генератора Галуа з чотирма розрядами(для прикладу). Регістри генераторів, розташовані вертикально та позначені символом "(x)", виконують операцію порозрядного множення, в той час як регістри, позначені символом "⊕", виконують операцію складання вмісту регістра за модулем 2.[1]

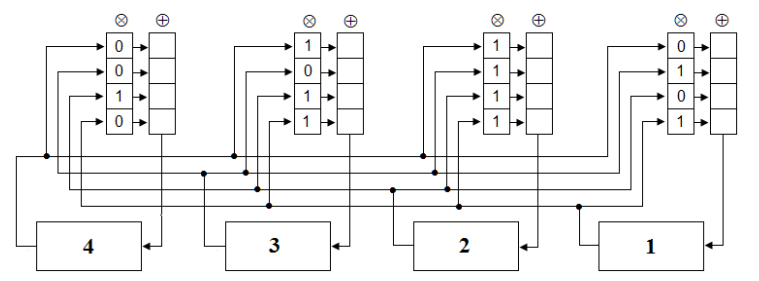


Рис. 1 Структурна схема узагальнених базових генераторів Псевдовипадкових послідовностей Галуа[[1]](#footnote-1)

Отже коротко складемо алгоритм роботи програми:

Матриця – ми будемо генерувати нашу узагальнену матрицю завдяки вхідним утворюючому та породжуючому поліному. Сама матриця тут буде основою генератора, через який буде проходити вектор ініціалізації та генерувати псевдовипадкове значення на виході.

Відповідно, згенерувавши, як описано на початку, узагальнену матрицю Галуа для нашого генератора нам потрібно прописати алгоритм самого генератора, через який і буде утворюватись псевдовипадковий елемент. Виходячи з представленого вище рисунку маємо – розкладаємо матрицю на колонки, тим самим утворюючи двійкові послідовності – старший розряд зверху, молодший – знизу. Блоки 1234 тут це вектор ініціалізації, який проходячи і утворюватиме наступний елемент послідовності. Отже, ми виконуємо побітове множення вектора на кожен стовпчик, утворюючи в нашому випадку 4 набори чисел. Після цього кожен набір ми просумовуємо за модулем 2, тобто XOR. Отже, з кожної колонки ми отримали по 1 біту наступного елементу, тепер записуємо новий елемент, ліва колонка – старший біт, права – молодший. Продовжуємо генерацію стільки разів, скільки потрібно, але в певний момент вона почне повторюватись. Для генератора 4розряди послідовність почне повторюватись через 2^4 елементи, тобто для загального 2^розрядність генератора, це максимум унікальності елементів послідовності для незвідного поліному.

Отже, побудуємо за алгоритмом програму на Python, яка генерує псевдовипадкові послідовності на основі теорії поля Галуа. Вона буде використовувати бібліотеку PyQt5 для створення графічного інтерфейсу користувача (GUI) для взаємодії з користувачем.[2, 3]

## Алгоритм роботи програми можна описати так:

### Імпортуємо необхідні модулі та бібліотеки:

* + модуль sys для системних параметрів і функцій.
  + Класи QApplication, QHeaderView, QMainWindow, QWidget, QLabel, QVBoxLayout, QLineEdit, QPushButton, QTableWidget, QTableWidgetItem і QMessageBox із модулів PyQt5.QtWidgets і PyQt5.QtGui для створення елементів GUI.[2]

### Визначимо глобальні змінні[3]:

* + result\_counter для відстеження поточної ітерації або результату.
  + result\_history для зберігання історії згенерованих результатів.
  + result\_table для збереження поточної таблиці результатів.
  + mainPolyPower для розрядності послідовності.

### Визначимо клас HistoryWindow[2, 3]:

* + Наслідує QMainWindow і представляє вікно для відображення історії згенерованих результатів.
  + Ініціалізує вікно та встановлює його властивості.
  + Визначає метод add\_history\_entry для додавання запису до таблиці історії.

### Визначимо клас ErrorWindow[2, 3]:

* + Наслідує QMessageBox і представляє вікно повідомлення про помилку.
  + Ініціалізує вікно та встановлює його властивості.

Визначимо клас MainWindow[2, 3]:

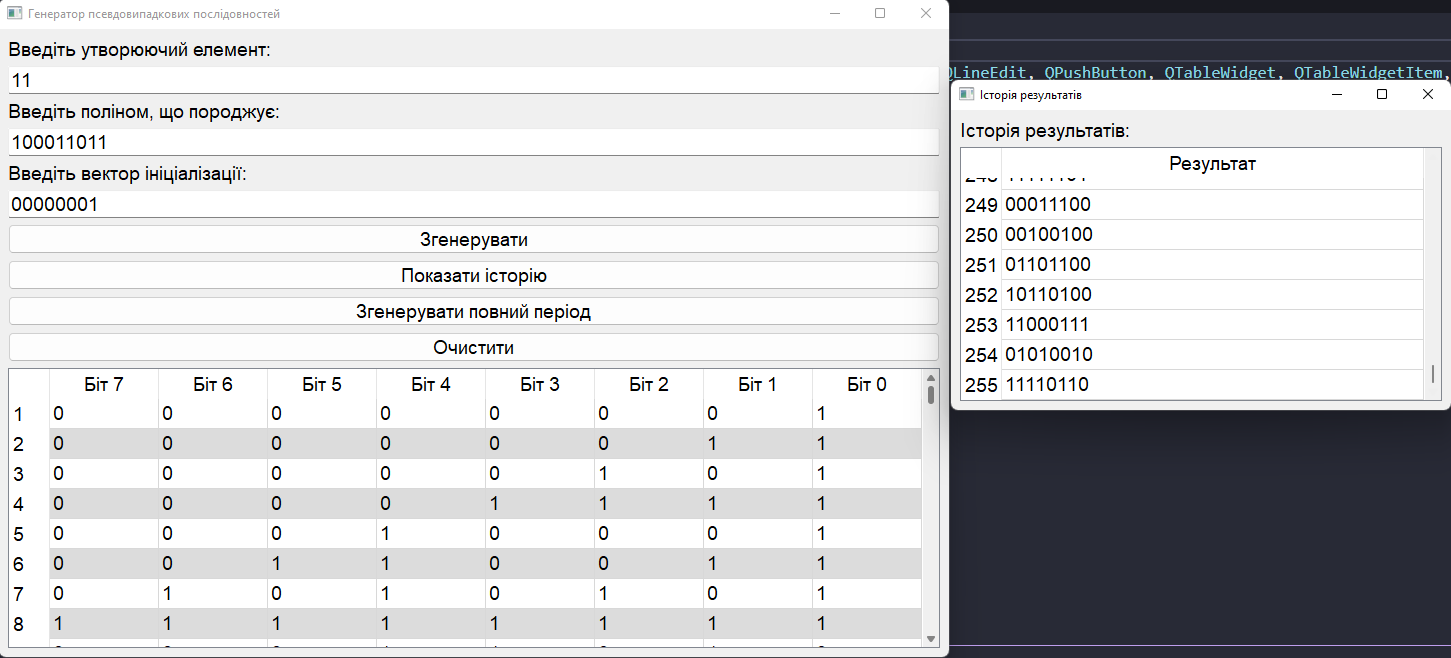
* + Наслідує QMainWindow і представляє головне вікно програми.
  + Ініціалізує вікно та встановлює його властивості.
  + Створює такі елементи графічного інтерфейсу, як мітки, поля введення тексту, кнопки та таблиці результатів.
  + Визначає методи генерації результатів, відображення історії результатів, генерації повної таблиці псевдовипадкової послідовності, очищення результатів і оновлення кнопки історії.
  + Використовує метод generate\_result для генерації псевдовипадкових послідовностей на основі даних користувача.
  + Використовує метод show\_result\_history для відображення історії згенерованих результатів в окремому вікні.
  + Використовує метод generate\_full\_period\_table для створення повної таблиці послідовності, доки не буде виявлено повторення.

### Визначимо основну точку входу програми [2]:

* + Створює екземпляр QApplication.
  + Створює екземпляр MainWindow.
  + Показує головне вікно.
  + Виконує цикл подій програми.
  + Алгоритм складається з серії кроків для перевірки введених користувачем даних, виконання обчислень на основі теорії поля Галуа, оновлення елементів GUI згенерованими результатами, обробки помилок і забезпечення взаємодії з користувачем за допомогою кнопок і полів введення.

[Перехід до лістингу](#Лістинг)

Зображення графічного інтерфейсу користувача:



# ВИСНОВКИ

В даній роботі було розглянуто синтез матриць на основі узагальненого правила діагонального заповнення. Була представлена структурна схема узагальнених базових генераторів Псевдовипадкових послідовностей Галуа. На основі цього було розроблено алгоритм і програму на мові програмування Python, яка генерує псевдовипадкові послідовності на основі теорії поля Галуа. Програма використовує бібліотеку PyQt5 для створення графічного інтерфейсу користувача (GUI) для взаємодії з користувачем.

Алгоритм роботи програми полягає у наступному:

1. Імпортування необхідних модулів та бібліотек.
2. Визначення глобальних змінних.
3. Визначення класу HistoryWindow для відображення історії згенерованих результатів.
4. Визначення класу ErrorWindow для вікна повідомлення про помилку.
5. Визначення класу MainWindow для головного вікна програми та його методів.
6. Основний алгоритм:
   * Створення узагальненої матриці Галуа.
   * Розкладання на стовпчики.
   * Перемноження, додавання по модулю 2.
   * Створення з цих даних елементу послідовності.
7. Визначення основної точки входу програми.

Отже, розроблено програму, яка здатна генерувати псевдовипадкові послідовності на основі узагальнених матриць Галуа. Це може бути корисно для застосування в криптографії, статистиці, тестуванні програмного забезпечення та багатьох інших галузях.

# ЛІТЕРАТУРА

1. [Примитивные матрицы галуа в криптографических приложениях, А. Я. Билецкий (https://www.researchgate.net).](https://www.researchgate.net)
2. [Програма використовує бібліотеку PyQt5 для створення графічного інтерфейсу користувача (GUI) і обробки взаємодії користувача. Документацію PyQt5 можна знайти за посиланням: https://www.riverbankcomputing.com/static/Docs/PyQt5/.](https://www.riverbankcomputing.com/static/Docs/PyQt5/)

1. [Додаткові модулі та функції Python, які використовуються в програмі, є частиною стандартної бібліотеки, і їх можна знайти в документації Python (https://docs.python.org/).](https://docs.python.org/)

# ДОДАТОК (Лістинг програми)

import sys

from PyQt5.QtWidgets import QApplication, QHeaderView, QMainWindow, QWidget, QLabel, QVBoxLayout, QLineEdit, QPushButton, QTableWidget, QTableWidgetItem, QMessageBox

from PyQt5.QtGui import QColor, QFont

result\_counter = 1

result\_history = []

result\_table = []

mainPolyPower = 2

class HistoryWindow(QMainWindow):

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

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

self.setWindowTitle("Історія результатів")

self.setGeometry(200, 200, 500, 300)

history\_widget = QWidget(self)

self.setCentralWidget(history\_widget)

history\_layout = QVBoxLayout(history\_widget)

history\_label = QLabel("Історія результатів:")

history\_layout.addWidget(history\_label)

self.history\_table = QTableWidget()

self.history\_table.setColumnCount(1)

self.history\_table.setHorizontalHeaderLabels(["Результат"])

self.history\_table.horizontalHeader().setSectionResizeMode(QHeaderView.Stretch)

history\_layout.addWidget(self.history\_table)

def add\_history\_entry(self, altBaseSeqPoly):

row\_count = self.history\_table.rowCount()

self.history\_table.setRowCount(row\_count + 1)

item\_alt\_poly = QTableWidgetItem(altBaseSeqPoly)

self.history\_table.setItem(row\_count, 0, item\_alt\_poly)

class ErrorWindow(QMessageBox):

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

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

self.setIcon(QMessageBox.Critical)

self.setWindowTitle("Помилка")

self.setText(message)

class MainWindow(QMainWindow):

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

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

self.setWindowTitle("Генератор псевдовипадкових послідовностей")

self.setGeometry(100, 100, 950, 500)

main\_widget = QWidget(self)

self.setCentralWidget(main\_widget)

main\_layout = QVBoxLayout(main\_widget)

input\_layout = QVBoxLayout()

main\_layout.addLayout(input\_layout)

galuaBasePoly\_label = QLabel("Введіть утворюючий елемент:")

input\_layout.addWidget(galuaBasePoly\_label)

self.galuaBasePoly\_entry = QLineEdit()

input\_layout.addWidget(self.galuaBasePoly\_entry)

mainSequencePoly\_label = QLabel("Введіть поліном, що породжує:")

input\_layout.addWidget(mainSequencePoly\_label)

self.mainSequencePoly\_entry = QLineEdit()

input\_layout.addWidget(self.mainSequencePoly\_entry)

altBaseSeqPoly\_label = QLabel("Введіть вектор ініціалізації:")

input\_layout.addWidget(altBaseSeqPoly\_label)

self.altBaseSeqPoly\_entry = QLineEdit()

input\_layout.addWidget(self.altBaseSeqPoly\_entry)

self.generate\_button = QPushButton("Згенерувати")

self.generate\_button.clicked.connect(self.generate\_result)

input\_layout.addWidget(self.generate\_button)

self.history\_button = QPushButton("Показати історію")

self.history\_button.clicked.connect(self.show\_result\_history)

input\_layout.addWidget(self.history\_button)

self.full\_period\_button = QPushButton("Згенерувати повний період")

self.full\_period\_button.clicked.connect(self.generate\_full\_period\_table)

input\_layout.addWidget(self.full\_period\_button)

self.clear\_button = QPushButton("Очистити")

self.clear\_button.clicked.connect(self.clear\_results)

input\_layout.addWidget(self.clear\_button)

self.result\_table = QTableWidget()

self.result\_table.setColumnCount(1)

self.result\_table.setHorizontalHeaderLabels(["Таблиця результатів"])

self.result\_table.horizontalHeader().setSectionResizeMode(QHeaderView.Stretch)

main\_layout.addWidget(self.result\_table)

self.history\_window = None

self.update\_history\_button()

# Установка шрифта и размера шрифта для всех виджетов

font = QFont("Arial", 14)

QApplication.instance().setFont(font)

self.setStyleSheet("QLabel, QLineEdit, QPushButton, QTableWidget { font-family: Arial; font-size: 14pt; }")

def generate\_result(self):

global result\_counter, result\_history, result\_table, mainPolyPower

galuaBasePoly = self.galuaBasePoly\_entry.text()

mainSequencePoly = self.mainSequencePoly\_entry.text()

altBaseSeqPoly = self.altBaseSeqPoly\_entry.text()

if not all(bit in ['0', '1'] for bit in galuaBasePoly) or not all(bit in ['0', '1'] for bit in mainSequencePoly):

self.clear\_results()

error\_message = "Дані введені не в двійковому форматі, спробуйте знову"

error\_window = ErrorWindow(error\_message)

error\_window.exec\_()

return

elif len(mainSequencePoly) <= len(galuaBasePoly) or len(mainSequencePoly) <= len(altBaseSeqPoly):

self.clear\_results()

error\_message = "Кількість бітів полінома, що породжує\nповинна бути більше кількості бітів УЕ і вектора ініціалізації"

error\_window = ErrorWindow(error\_message)

error\_window.exec\_()

return

try:

galuaBasePoly\_dec = int(galuaBasePoly, 2)

mainSequencePoly\_dec = int(mainSequencePoly, 2)

tableArr = []

tableBinNum = galuaBasePoly\_dec

mainPolyPower = len(mainSequencePoly)

if len(result\_table) >= (2 \*\* (mainPolyPower - 1))-1:

self.clear\_results()

for i in range(mainPolyPower - 1):

binary\_str = format(tableBinNum, f'0{mainPolyPower - 1}b')

tableArr.append(binary\_str)

tableBinNum = tableBinNum << 1

if tableBinNum > 2 \*\* (mainPolyPower - 1):

tableBinNum = tableBinNum ^ mainSequencePoly\_dec

tableArr.reverse()

self.result\_table.clear()

self.result\_table.setRowCount(len(tableArr))

for i, binary\_str in enumerate(tableArr):

item = QTableWidgetItem(binary\_str)

self.result\_table.setItem(i, 0, item)

if altBaseSeqPoly == '':

altBaseSeqPoly = format(1, f'0{mainPolyPower - 1}b')

else:

altBaseSeqPoly = format(int(altBaseSeqPoly[:mainPolyPower - 1], 2), f'0{mainPolyPower - 1}b')

result = []

for i in range(mainPolyPower - 1):

column\_result = 0

for j in range(len(tableArr)):

column\_result ^= int(tableArr[j][i], 2) & int(altBaseSeqPoly[j], 2)

result.append(column\_result)

result\_bin = ''.join(str(bit) for bit in result)

result\_table.append(altBaseSeqPoly) # добавляем значение в таблицу

self.generate\_period\_table()

result\_history.append(altBaseSeqPoly)

self.update\_history\_button()

self.altBaseSeqPoly\_entry.clear()

self.altBaseSeqPoly\_entry.insert(result\_bin)

if self.history\_window is not None:

self.history\_window.add\_history\_entry(altBaseSeqPoly)

if result\_counter == 2:

self.generate\_button.setText("Наступна ітерація")

except ValueError as e:

self.result\_table.clear()

self.result\_table.setRowCount(0)

self.result\_table.insertRow(0)

self.result\_table.setItem(0, 0, QTableWidgetItem("Помилка: " + str(e)))

def show\_result\_history(self):

self.update\_history\_button()

if self.history\_window is None:

self.history\_window = HistoryWindow()

history\_table = self.history\_window.history\_table

history\_table.setColumnCount(1)

history\_table.setHorizontalHeaderLabels(["Результат"])

history\_table.horizontalHeader().setSectionResizeMode(QHeaderView.Stretch)

for i, result in enumerate(result\_history, start=1):

history\_table.insertRow(i - 1)

history\_table.setItem(i - 1, 0, QTableWidgetItem(result))

self.history\_window.show()

def clear\_results(self):

global result\_counter, result\_table, mainPolyPower

result\_counter = 1

result\_table = []

self.result\_table.clear()

self.result\_table.setRowCount(0) # Залишаємо один рядок для виводу помилки

self.result\_table.setColumnCount(0)

def update\_history\_button(self):

has\_history = len(result\_history) > 0

self.history\_button.setEnabled(has\_history)

def generate\_period\_table(self):

self.result\_table.clearContents()

self.result\_table.setColumnCount(len(result\_table[0])) # Set the number of columns based on the length of binary strings

self.result\_table.setHorizontalHeaderLabels([f"Біт {i}" for i in range(len(result\_table[0])-1, -1, -1)])

row\_count = len(result\_table) # Adjust the row count based on the number of results

self.result\_table.setRowCount(row\_count)

for i, binary\_str in enumerate(result\_table):

for j, bit in enumerate(binary\_str):

item = QTableWidgetItem(bit)

self.result\_table.setItem(i, j, item)

# Set background color for every second row

if i % 2 == 1:

item.setBackground(QColor(220, 220, 220)) # Light gray color

def generate\_full\_period\_table(self):

galuaBasePoly = self.galuaBasePoly\_entry.text()

mainSequencePoly = self.mainSequencePoly\_entry.text()

altBaseSeqPoly = self.altBaseSeqPoly\_entry.text()

if not all(bit in ['0', '1'] for bit in galuaBasePoly) or not all(bit in ['0', '1'] for bit in mainSequencePoly):

self.clear\_results()

error\_message = "Дані введені не в двійковому форматі, спробуйте знову"

error\_window = ErrorWindow(error\_message)

error\_window.exec\_()

return

elif len(mainSequencePoly) <= len(galuaBasePoly) or len(mainSequencePoly) <= len(altBaseSeqPoly):

self.clear\_results()

error\_message = "Кількість бітів полінома, що породжує\nповинна бути більше кількості бітів УЕ і вектора ініціалізації"

error\_window = ErrorWindow(error\_message)

error\_window.exec\_()

return

while len(result\_table) == len(set(result\_table)) and len(result\_table) < (2 \*\* (mainPolyPower - 1))-1:

self.generate\_result()

if \_\_name\_\_ == '\_\_main\_\_':

app = QApplication(sys.argv)

window = MainWindow()

window.show()

sys.exit(app.exec\_())

1. Примитивные матрицы галуа в криптографических приложениях, А. Я. Билецкий (<https://www.researchgate.net>) [↑](#footnote-ref-1)