МИНИСТЕРСТВО ОБРАЗОВАНИЯ РЕСПУБЛИКИ БЕЛАРУСЬ
УЧРЕЖДЕНИЕ ОБРАЗОВАНИЯ
«БРЕСТСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ»
ФАКУЛЬТЕТ ЭЛЕКТРОННО-ИНФОРМАЦИОННЫХ СИСТЕМ
Кафедра интеллектуальных информационных технологий

Отчет
по дисциплине
«Современные методы защиты информации»
по лабораторной работе № 4
«Сумматор в квантовых схемах»

Выполнила:
студентка 4
курса группы
ИИ-22
Сокол С.М.
Проверила:
Хацкевич А.С.

Брест 2024

Цель: ознакомление с выполнением простого сложения с помощью квантовых схем.

Постановка задачи:

- Изучить теоретический материал.
- Средствами Qiskit или используя средства интерактивной среды IBM Quantum Experience, создать квантовую схему полного сумматора.

A(input)	B(input)	X(carry input)	S(sum)	C(carry out)
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

Ход работы:

```
from qiskit import QuantumRegister, ClassicalRegister, QuantumCircuit
from qiskit_aer import Aer
import matplotlib
import matplotlib.pyplot as plt

matplotlib.use('QtAgg')

def get_result(a: int, b: int, x: int, shots_count: int = 1024, test: bool =
False) -> dict:
    # Определяем квантовые регистры
    a_reg = QuantumRegister(1, 'a')
    b_reg = QuantumRegister(1, 'b')
    x_reg = QuantumRegister(1, 'x')
    qt = QuantumRegister(6, 'temp')  # Временные регистры

    cr = ClassicalRegister(2, 'c')  # Классический регистр для измерений

    # Создаем квантовую цепь
    qc = QuantumCircuit(a_reg, b_reg, x_reg, qt, cr)

    # Инициализируем кубиты в зависимости от входных значений
    if test:
        qc.h(a_reg[0])
        qc.h(b_reg[0])
        qc.h(x_reg[0])
    else:
        if a:
            qc.x(a_reg[0])  # Устанавливаем A
        if b:
```

```
            qc.x(b_reg[0])  # Устанавливаем B
        if x:
            qc.x(x_reg[0])  # Устанавливаем X

    # Квантовые операции
    qc.cx(a_reg[0], qt[0])  # A -> T[0]
    qc.cx(b_reg[0], qt[0])  # A xor B -> T[0]
    qc.cx(x_reg[0], qt[0])  # A xor B xor X -> T[0]
    qc.measure(qt[0], cr[0])  # Измеряем T[0] -> C[0]

    # Операции CCNOT для вычисления AND
    qc.ccx(a_reg[0], b_reg[0], qt[1])  # A and B -> T[1]
    qc.ccx(a_reg[0], x_reg[0], qt[2])  # A and X -> T[2]
    qc.ccx(b_reg[0], x_reg[0], qt[3])  # B and X -> T[3]

    # Объединяем результаты с помощью CNOT и CCNOT
    qc.cx(qt[2], qt[4])
    qc.cx(qt[1], qt[4])
    qc.ccx(qt[1], qt[2], qt[4])  # (A and B) or (A and X) -> T[4]

    qc.cx(qt[4], qt[5])
    qc.cx(qt[3], qt[5])
    qc.ccx(qt[3], qt[4], qt[5])  # (A and B) or (A and X) or (B and X) -> T[5]

    qc.measure(qt[5], cr[1])  # Измеряем T[5] -> C[1]

    # Выполняем цепь с использованием симулятора Aer
    simulator = Aer.get_backend('aer_simulator')
    result = simulator.run(qc, shots=shots_count).result()
    counts = result.get_counts(qc)

    # Визуализируем цепь, если в тестовом режиме
    if test:
        qc.draw(output='mpl')
        plt.show()  # Выводим график

    return counts

def main():
    shots = 10000
    temp = get_result(0, 0, 0, shots, True)

    print(f"{' A':>3s}|{' B':>3s}|{' X':>3s}||{' S':>3s}|{' C':>3s}|")
    print(f"{'---':>3s}|{'---':>3s}|{'---':>3s}||{'---':>3s}|{'---':>3s}|")

    for i in range(8):
        # Извлекаем входные аргументы из индекса итерации
```

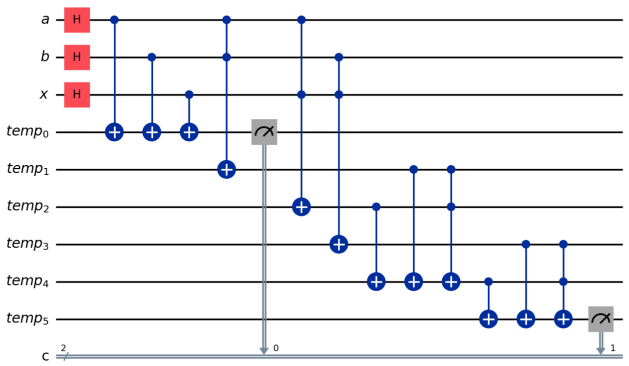
```
        arg_a = (i & 0x4) >> 2
        arg_b = (i & 0x2) >> 1
        arg_x = i & 0x1

        res = get_result(arg_a, arg_b, arg_x, 1)

        t = list(res.keys())[0]
        s = t[0]  # Результат S
        c = t[1]  # Результат C

        print(f" {str(arg_a):>2s}| {str(arg_b):>2s}| {str(arg_x):>2s}||
{s:>2s}| {c:>2s}|  {(temp[t] / shots):1.4f}")

if __name__ == "__main__":
    main()
```



A	B	X	S	C	
0	0	0	0	0	0.1239
0	0	1	0	1	0.3772
0	1	0	0	1	0.3772
0	1	1	1	0	0.3705
1	0	0	0	1	0.3772
1	0	1	1	0	0.3705
1	1	0	1	0	0.3705
1	1	1	1	1	0.1284

Вывод: ознакомилась с выполнением простого сложения с помощью квантовых схем.