A Novel Multi-User Quantum Communication System Using CDMA and Quantum Fourier Transform

Motivation:

1. Data transfers from source to destination via untrusted Router/Switch
2. Chance of eavesdropping

3. To restrict provide 2 layers security
   - 3.1: Instead of classical bits send quantum bits (1st layer security)
   - 3.2: Then Multiplex and Demultiplex the encoded qubit . But How?
   - 3.3: Apply Unitary Transformation(QFT*IQFT=Identity) to increase 2nd layer security

Increasing the difficulty for eavesdropper to guess the Qubit being exchanged across routers.

As a result: eavesdropping become very very difficult

Soumen Das
2025BSZ8209

A Novel Multi-User Quantum Communication System Using CDMA and Quantum Fourier Transform

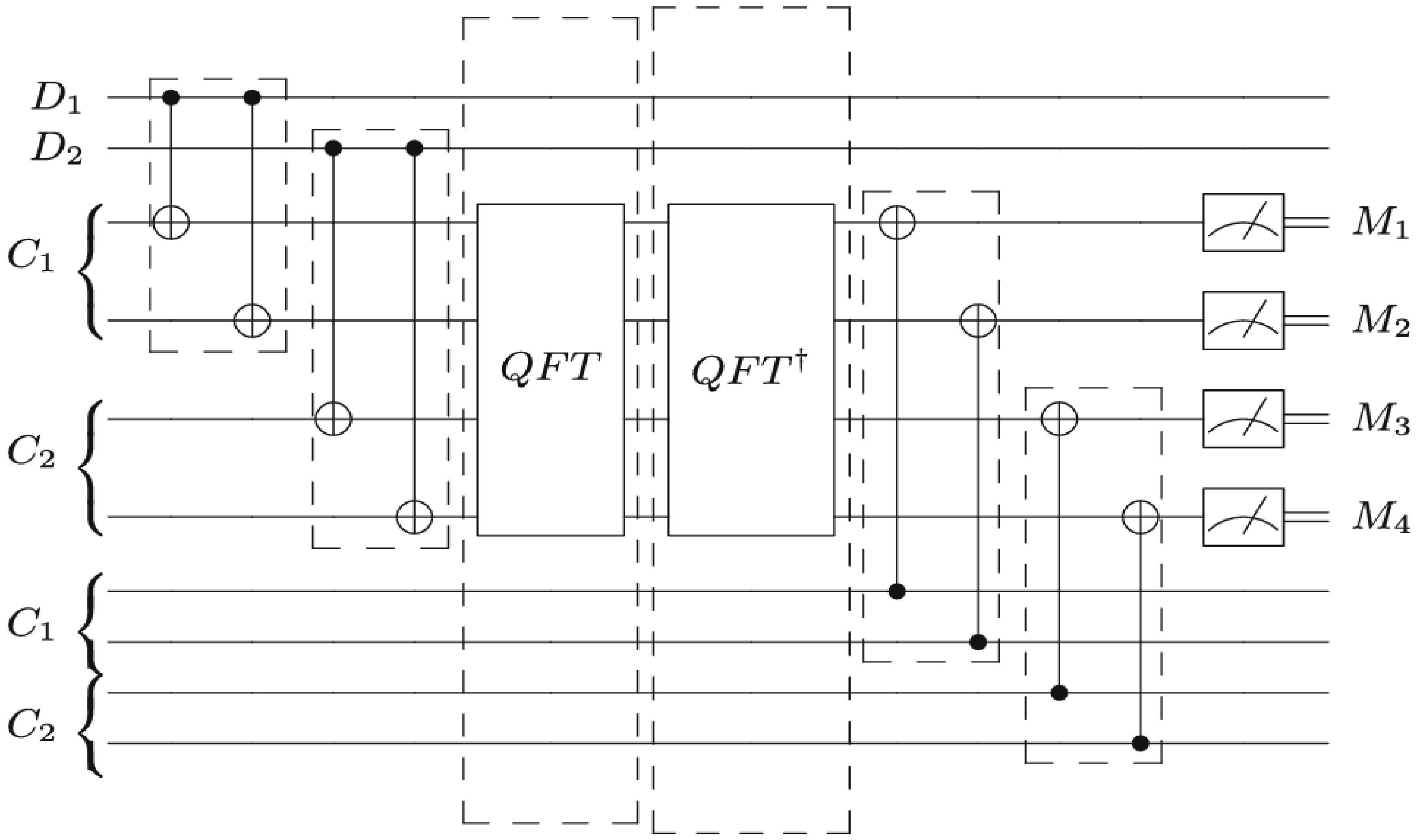


Fig. 2. Proposed Multi-User Quantum circuit.

STEP 1

User $_1$: classical bit $= d_1$ (corresponding **qubit** $= D_1$)
User $_2$: classical bit $= d_2$ (corresponding **qubit** $= D_2$)

$$E_1 = |E_{1x}E_{1y}\rangle$$
$$E_{1x} = C_{1x} \oplus D_1$$
$$E_{1y} = C_{1y} \oplus D_1$$

$$E_2 = |E_{2x}E_{2y}\rangle$$
$$E_{2x} = C_{2x} \oplus D_2$$
$$E_{2y} = C_{2y} \oplus D_2$$

CDMA Encoded data for both Users

$$E = |E_{1x}E_{1y}\, E_{2x}E_{2y}\rangle = |D_1\, D_1'\, D_2 D_2\rangle$$

User $_1$: Encoding

$|D_1\rangle = |d_1\rangle = \{0,1\}$, $C_1 = |C_{1x}\, C_{1y}\rangle = |01\rangle$

If $D_1=0$, $E_{1x} = C_{1x} \oplus D_1 = 0 \oplus 0 = 0$

If $D_1=1$, $E_{1x} = C_{1x} \oplus D_1 = 0 \oplus 1 = 1$ $\quad$ **$E_{1x} = D_1$**

If $D_1=0$, $E_{1y} = C_{1y} \oplus D_1 = 1 \oplus 0 = 1$

If $D_1=1$, $E_{1y} = C_{1y} \oplus D_1 = 1 \oplus 1 = 0$ $\quad$ **$E_{1y} = D_1'$ (complement of D_1)**

$$E_1 = |E_{1x}\, E_{1y}\rangle = |D_1, 1 \oplus D_1\rangle = |D_1, D_1'\rangle$$

User $_2$: Encoding

$|D_2\rangle = |d_2\rangle = \{0,1\}$, $C_2 = |C_{2x}\, C_{2y}\rangle = |00\rangle$

If $D_2=0$, $E_{2x} = C_{2x} \oplus D_2 = 0 \oplus 0 = 0$

If $D_2=1$, $E_{2x} = C_{2x} \oplus D_2 = 0 \oplus 1 = 1$ $\quad$ **$E_{2x} = D_2$**

If $D_2=0$, $E_{2y} = C_{2y} \oplus D_2 = 1 \oplus 0 = 1$

If $D_2=1$, $E_{2y} = C_{2y} \oplus D_2 = 1 \oplus 1 = 0$ $\quad$ **$E_{2y} = D_2$**

$$E_2 = |E_{2x}\, E_{2y}\rangle = |D_2, D_2\rangle$$

CDMA encoded data for both Users

$$E = |E_{1x}E_{1y}\, E_{2x}E_{2y}\rangle = |D_1\, D_1'\, D_2\, D_2\rangle$$

STEP 2

Code state before QFT

After CDMA Encoding $(C_{1x}, C_{1y}, C_{2x}, C_{2y})$,

$$|C'\rangle = |d_1, 1 \oplus d_1, d_2, d_2\rangle.$$

Interpret this 4-bit string as an integer x with the leftmost bit most-significant:

$$x = d_1.2^3 + (1 \oplus d_1).2^2 + d_2.2^1 + d_2.2^0$$

Simplify:

$$x = 4d_1 + 3d_2 + 4.$$

d_1, d_2	$x = d_1.2^3 + (1 \oplus d_1).2^2 + d_2.2^1 + d_2.2^0$
00	4
01	7
10	8
11	11

Input to QFT Block $x = \{4, 7, 8, 11\}$

QFT on a 4-qubit basis state

For $n = 4$ qubits

$$\text{QFT } |x\rangle = \frac{1}{\sqrt{16}} \sum_{k=0}^{15} e^{2\pi i \frac{xk}{16}} \, |k\rangle = \frac{1}{4} \sum_{k=0}^{15} e^{2\pi i xk/16} \, |k\rangle.$$

Full state after the QFT (on the code register only) is

$$|\Psi_{\text{after QFT}}\rangle = |d_1 d_2\rangle \otimes \frac{1}{4} \sum_{k=0}^{15} e^{\frac{2\pi i xk}{16}} \, |k\rangle,$$

STEP 3

How the (IQFT) applied to the QFT output collapses to $|x\rangle$ for each $x \in \{4, 7, 8, 11\}$

- Number of qubits $n = 4$, so $N = 2^n = 16$.
- QFT on basis state $|x\rangle$:

$$\text{QFT}\ |x\rangle = \frac{1}{4}\sum_{k=0}^{15} e^{\frac{2\pi i}{16}xk}\ |k\rangle.$$

- IQFT on basis state $|k\rangle$:

$$\text{IQFT}\ |k\rangle = \frac{1}{4}\sum_{j=0}^{15} e^{-\frac{2\pi i}{16}jk}\ |j\rangle.$$

- compute IQFT(QFT $|x\rangle$)
- $$\text{IQFT}(\text{QFT}\,|x\rangle) = \frac{1}{16}\sum_{j=0}^{15}\left(\sum_{k=0}^{15} e^{\frac{2\pi i}{16}(x-j)k}\right)\ |j\rangle$$

Define $m := x - j$. The inner sum is

$$S_m := \sum_{k=0}^{15} e^{\frac{2\pi i}{16}mk}.$$

For integer m,

$$S_m = \sum_{k=0}^{15}\left(e^{\frac{2\pi i}{16}m}\right)^k$$

$$= \frac{1-\left(e^{\frac{2\pi i}{16}m}\right)^{16}}{1-e^{\frac{2\pi i}{16}m}}\ \text{(GP, valid if Denominator} \neq 0\text{)}.$$

But $\left(e^{\frac{2\pi i}{16}m}\right)^{16} = e^{2\pi i m} = 1$ for any integer m.

The Numerator $1 - 1 = 0$.

Case 1

- If the Denominator $1 - e^{\frac{2\pi i}{16}m} \neq 0$(i.e. $e^{\frac{2\pi i}{16}m} \neq 1$), then $S_m = 0/(\text{nonzero}) = 0$.

Case 2

- If the Denominator $1 - e^{\frac{2\pi i}{16}m} = 0$, means $e^{\frac{2\pi i}{16}m} = 1$.
- That occurs exactly when $\frac{2\pi}{16}m$ is an integer multiple of 2π, i.e. $m \equiv 0 \pmod{16}$. In that case every term in the sum equals 1, so $S_m = 16$.
-

So compactly:

$$\boxed{S_m = \sum_{k=0}^{15} e^{\frac{2\pi i}{16}mk} = \begin{cases} 16 & m \equiv 0 \pmod{16} \\ 0 & \text{otherwise.} \end{cases}}$$

STEP 4

Case 1 — $x = 4$

$$\text{QFT}\mid 4\rangle = \frac{1}{4}\sum_{k=0}^{15} e^{\frac{2\pi i}{16}4k}\mid k\rangle.$$

Apply IQFT:

$$\text{IQFT}(\text{QFT}\mid 4\rangle) = \frac{1}{16}\sum_{j=0}^{15}\left(\sum_{k=0}^{15} e^{\frac{2\pi i}{16}(4-j)k}\right)\mid j\rangle$$

$$= \frac{1}{16}\sum_{j=0}^{15} S_{4-j}\mid j\rangle.$$

For each j evaluate $m = 4 - j$

- **If $j = 4$ then $m = 0$ so $S_0 = 16$.**
- **For any $j \neq 4$, m is not divisible by 16, so $S_m = 0$.**
- **In other words, $4 - j = 0 (mod\ 16)$**

Therefore, the sum collapses:

$$\text{IQFT}(\text{QFT}\mid 4\rangle) = \frac{1}{16}(16\mid 4\rangle) = \mid 4\rangle.$$

Case 2 — $x = 7$

$$\text{QFT}\mid 7\rangle = \frac{1}{4}\sum_{k=0}^{15} e^{\frac{2\pi i}{16}7k}\mid k\rangle.$$

Apply IQFT:

$$\text{IQFT}(\text{QFT}\mid 7\rangle) = \frac{1}{16}\sum_{j=0}^{15}\left(\sum_{k=0}^{15} e^{\frac{2\pi i}{16}(7-j)k}\right)\mid j\rangle$$

$$= \frac{1}{16}\sum_{j=0}^{15} S_{7-j}\mid j\rangle.$$

For each j evaluate $m = 7 - j$

- **If $j = 7$ then $m = 0$ so $S_0 = 16$.**
- **For any $j \neq 7$, m is not divisible by 16, so $S_m = 0$.**
- **In other words, $7 - j = 0 (mod\ 16)$**

Therefore, the sum collapses:

$$\text{IQFT}(\text{QFT}\mid 7\rangle) = \frac{1}{16}(16\mid 7\rangle) = \mid 7\rangle.$$

Case 3 — $x = 8$

$$\text{QFT} \mid 8\rangle = \frac{1}{4}\sum_{k=0}^{15} e^{\frac{2\pi i}{16}8k} \mid k\rangle.$$

Apply IQFT:

$$\text{IQFT}(\text{QFT} \mid 8\rangle) = \frac{1}{16}\sum_{j=0}^{15}\left(\sum_{k=0}^{15} e^{\frac{2\pi i}{16}(8-j)k}\right) \mid j\rangle$$

$$= \frac{1}{16}\sum_{j=0}^{15} S_{8-j} \mid j\rangle.$$

For each j evaluate $m = 8 - j$

- If $j = 8$ then $m = 0$ so $S_0 = 16$.
- For any $j \neq 8$, m is not divisible by 16, so $S_m = 0$.
- In other words, $8 - j = 0(mod\ 16)$

Therefore, the sum collapses:

$$\text{IQFT}(\text{QFT} \mid 8\rangle) = \frac{1}{16}(16 \mid 8\rangle) = \mid 8\rangle.$$

Case 4 — $x = 11$

$$\text{QFT} \mid 11\rangle = \frac{1}{4}\sum_{k=0}^{15} e^{\frac{2\pi i}{16}11k} \mid k\rangle.$$

Apply IQFT:

$$\text{IQFT}(\text{QFT} \mid 11\rangle) = \frac{1}{16}\sum_{j=0}^{15}\left(\sum_{k=0}^{15} e^{\frac{2\pi i}{16}(11-j)k}\right) \mid j\rangle$$

$$= \frac{1}{16}\sum_{j=0}^{15} S_{11-j} \mid j\rangle.$$

For each j evaluate $m = 11 - j$

- If $j = 11$ then $m = 0$ so $S_0 = 16$.
- For any $j \neq 11$, m is not divisible by 16, so $S_m = 0$.
- In other words, $11 - j = 0(mod\ 16)$

Therefore, the sum collapses:

$$\text{IQFT}(\text{QFT} \mid 11\rangle) = \frac{1}{16}(16 \mid 11\rangle) = \mid 11\rangle.$$

STEP 5

The CDMA-Decoded qubits are

$$F = | F_{1x}\ F_{1y}\ F_{2x}\ F_{2y} \rangle$$

$$\begin{aligned} F_{1x} &= C_{1x} \oplus E_{1x}, & M_1 &= \text{Measure}(F_{1x}) \\ F_{1y} &= C_{1y} \oplus E_{1y}, & M_2 &= \text{Measure}(F_{1y}) \\ F_{2x} &= C_{2x} \oplus E_{2x}, & M_3 &= \text{Measure}(F_{2x}) \\ F_{2y} &= C_{2y} \oplus E_{2y}. & M_4 &= \text{Measure}(F_{2y}) \end{aligned}$$

Recovery rule:

$$d_1 = M_1 \wedge M_2, d_2 = M_3 \wedge M_4$$

Substitute the known C and E

Plug C and E into the F definitions:

1. $F_{1x} = C_{1x} \oplus E_{1x} = 0 \oplus d_1 = d_1$.
2. $F_{1y} = C_{1y} \oplus E_{1y} = 1 \oplus (1 \oplus d_1) = d_1$

   Use XOR associativity and $a \oplus a = 0$:

   $$1 \oplus (1 \oplus d_1) = (1 \oplus 1) \oplus d_1 = 0 \oplus d_1 = d_1.$$

   So $F_{1y} = d_1$.
3. $F_{2x} = C_{2x} \oplus E_{2x} = 0 \oplus d_2 = d_2$.
4. $F_{2y} = C_{2y} \oplus E_{2y} = 0 \oplus d_2 = d_2$.

So, the four Decoded qubits simplify to

$$F = | d_1\ d_1\ d_2\ d_2 \rangle.$$

STEP 6

Measurement and final recovery

Measuring F gives

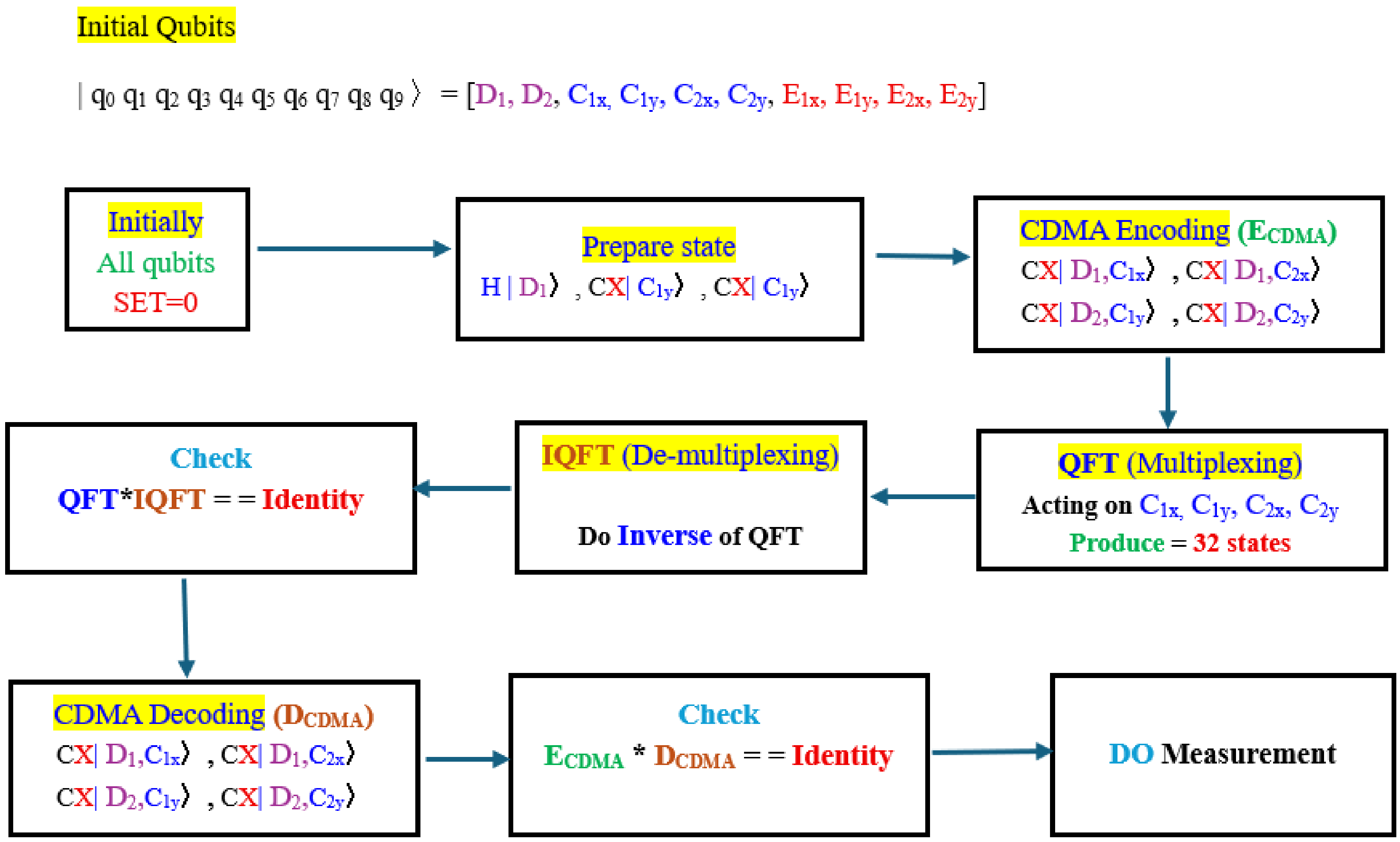
$$(M_1, M_2, M_3, M_4) = (d_1, d_1, d_2, d_2)$$

Apply the recovery rule:

- $d_1^{(\text{recovered})} = M_1 \wedge M_2 = d_1 \wedge d_1 = d_1$
- $d_2^{(\text{recovered})} = M_3 \wedge M_4 = d_2 \wedge d_2 = d_2$

original bits are recovered exactly.

(d1,d2)	F=(F1x,F1y,F2x,F2y)	M=(M1,M2,M3,M4)	Recovered d_1, d_2
(0,0)	(0,0,0,0)	(0,0,0,0)	(0,0)
(0,1)	(0,0,1,1)	(0,0,1,1)	(0,1)
(1,0)	(1,1,0,0)	(1,1,0,0)	(1,0)
(1,1)	(1,1,1,1)	(1,1,1,1)	(1,1)



Initial Qubits
| q0 q1 q2 q3 q4 q5 q6 q7 q8 q9 〉 = [D1, D2, C1x, C1y, C2x, C2y, E1x, E1y, E2x, E2y]
Initially
All qubits
SET=0
Prepare state
H | D1〉 , CX| C1y〉 , CX| C1y〉
CDMA Encoding (ECDMA)
CX| D1,C1x〉 , CX| D1,C2x〉
CX| D2,C1y〉 , CX| D2,C2y〉
QFT (Multiplexing)
Acting on C1x, C1y, C2x, C2y
Produce = 32 states
IQFT (De-multiplexing)
Do Inverse of QFT
Check
QFT*IQFT = = Identity
CDMA Decoding (DCDMA)
CX| D1,C1x〉 , CX| D1,C2x〉
CX| D2,C1y〉 , CX| D2,C2y〉
Check
ECDMA * DCDMA = = Identity
DO Measurement

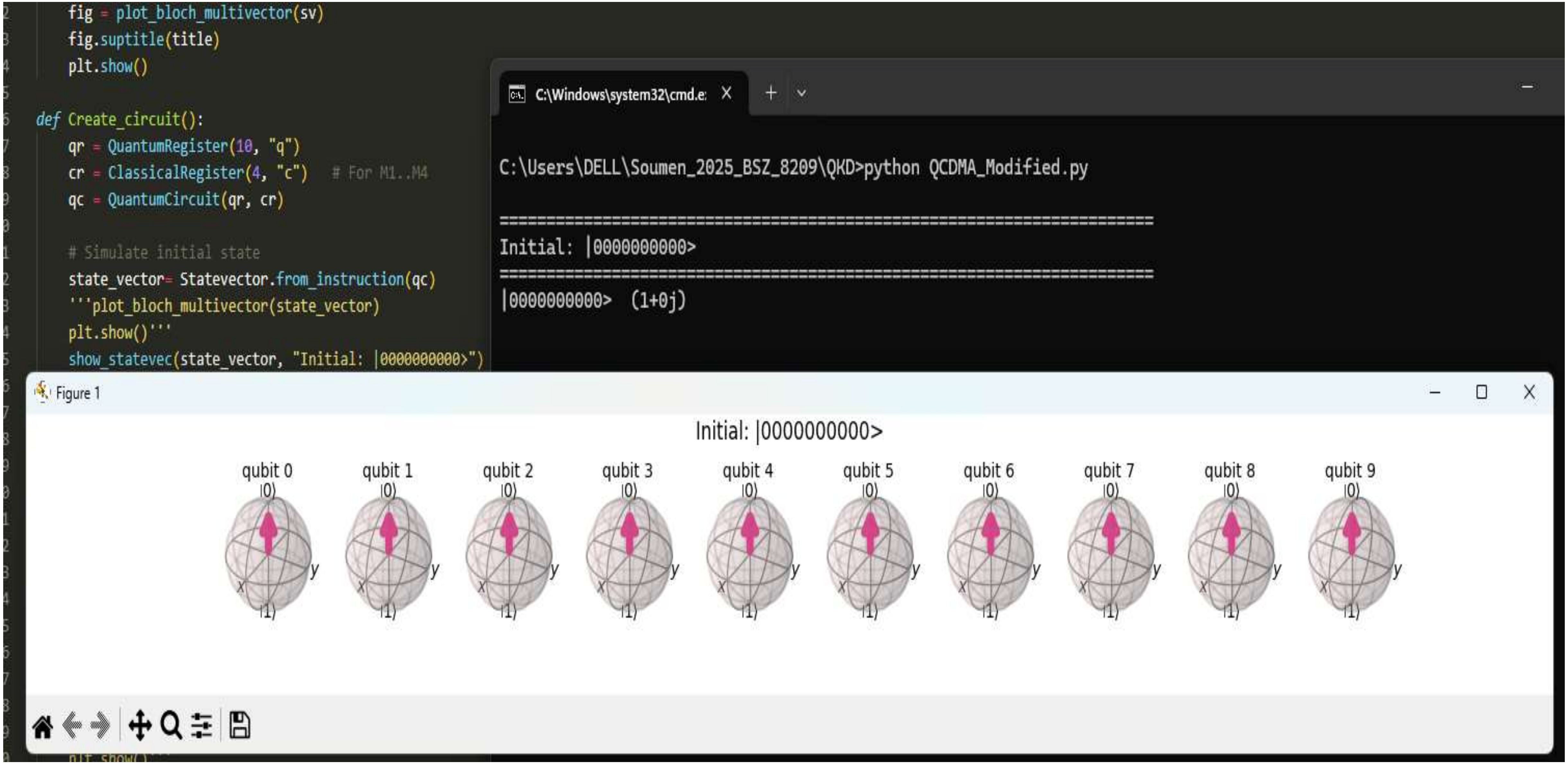
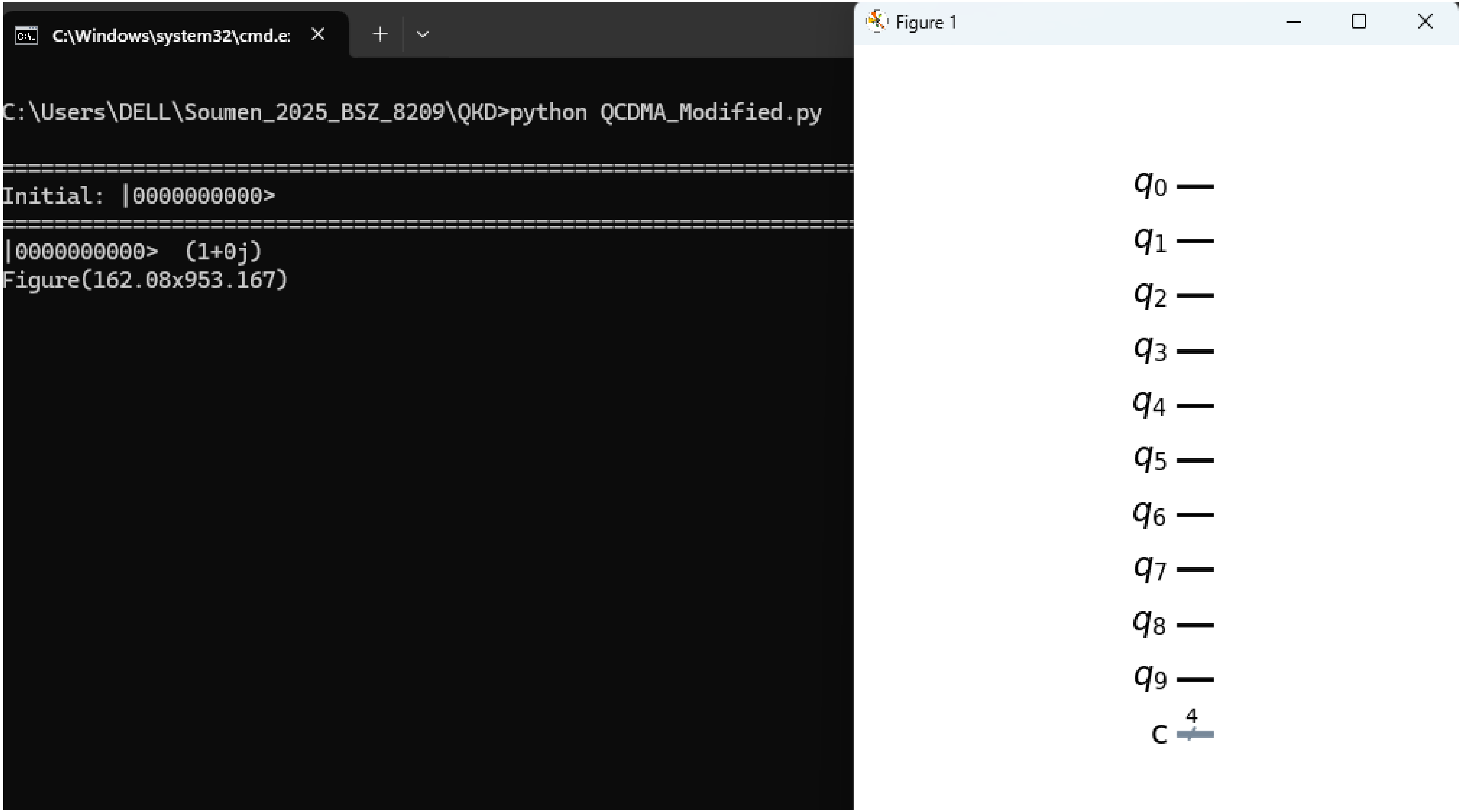


fig = plot_bloch_multivector(sv)
fig.suptitle(title)
plt.show()
def Create_circuit():
qr = QuantumRegister(10, "q")
cr = ClassicalRegister(4, "c")   # For M1..M4
qc = QuantumCircuit(qr, cr)
# Simulate initial state
state_vector= Statevector.from_instruction(qc)
'''plot_bloch_multivector(state_vector)
plt.show()'''
show_statevec(state_vector, "Initial: |0000000000>")
C:\Windows\system32\cmd.e:
C:\Users\DELL\Soumen_2025_BSZ_8209\QKD>python QCDMA_Modified.py
==
Initial: |0000000000>
==
|0000000000> (1+0j)
Figure 1
Initial: |0000000000>
qubit 0
qubit 1
qubit 2
qubit 3
qubit 4
qubit 5
qubit 6
qubit 7
qubit 8
qubit 9

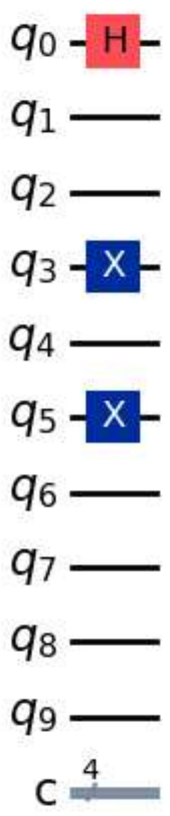

C:\Windows\system32\cmd.e:
C:\Users\DELL\Soumen_2025_BSZ_8209\QKD>python QCDMA_Modified.py
Initial: |0000000000>
|0000000000> (1+0j)
Figure(162.08x953.167)
Figure 1
q_0
q_1
q_2
q_3
q_4
q_5
q_6
q_7
q_8
q_9
c
4

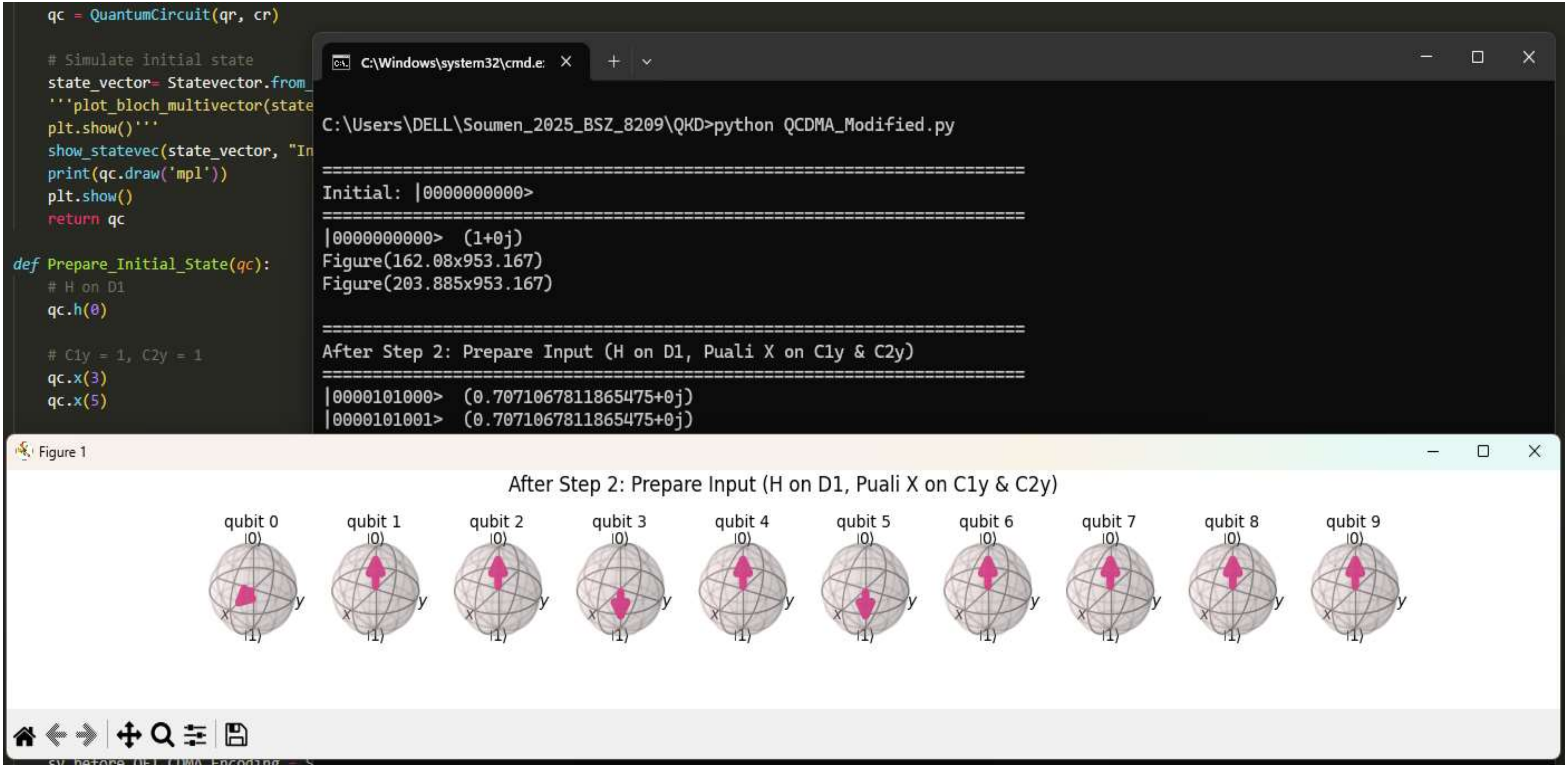
```python
    # Simulate initial state
    state_vector= Statevector.from_instruction(qc)
    '''plot_bloch_multivector(state
    plt.show()'''
    show_statevec(state_vector, "In
    print(qc.draw('mpl'))
    plt.show()
    return qc

def Prepare_Initial_State(qc):
    # H on D1
    qc.h(0)

    # C1y = 1, C2y = 1
    qc.x(3)
    qc.x(5)

    state_vector_original= Statevec
    '''plot_bloch_multivector(state
    plt.show()'''
    print(qc.draw('mpl'))
    plt.show()
    show_statevec(state_vector_orig
    return qc,state_vector_original
    pass

def CDMA_Encoding(qc):
    qc.cx(0, 2)
    qc.cx(0, 4)
    qc.cx(1, 3)
    qc.cx(1, 5)
    sv_before_QFT_CDMA_Encoding = S
    show_statevec(sv_before_QFT_CDM
    print(qc.draw('mpl'))
    plt.show()
    return qc,sv_before_QFT_CDMA_Encoding
```

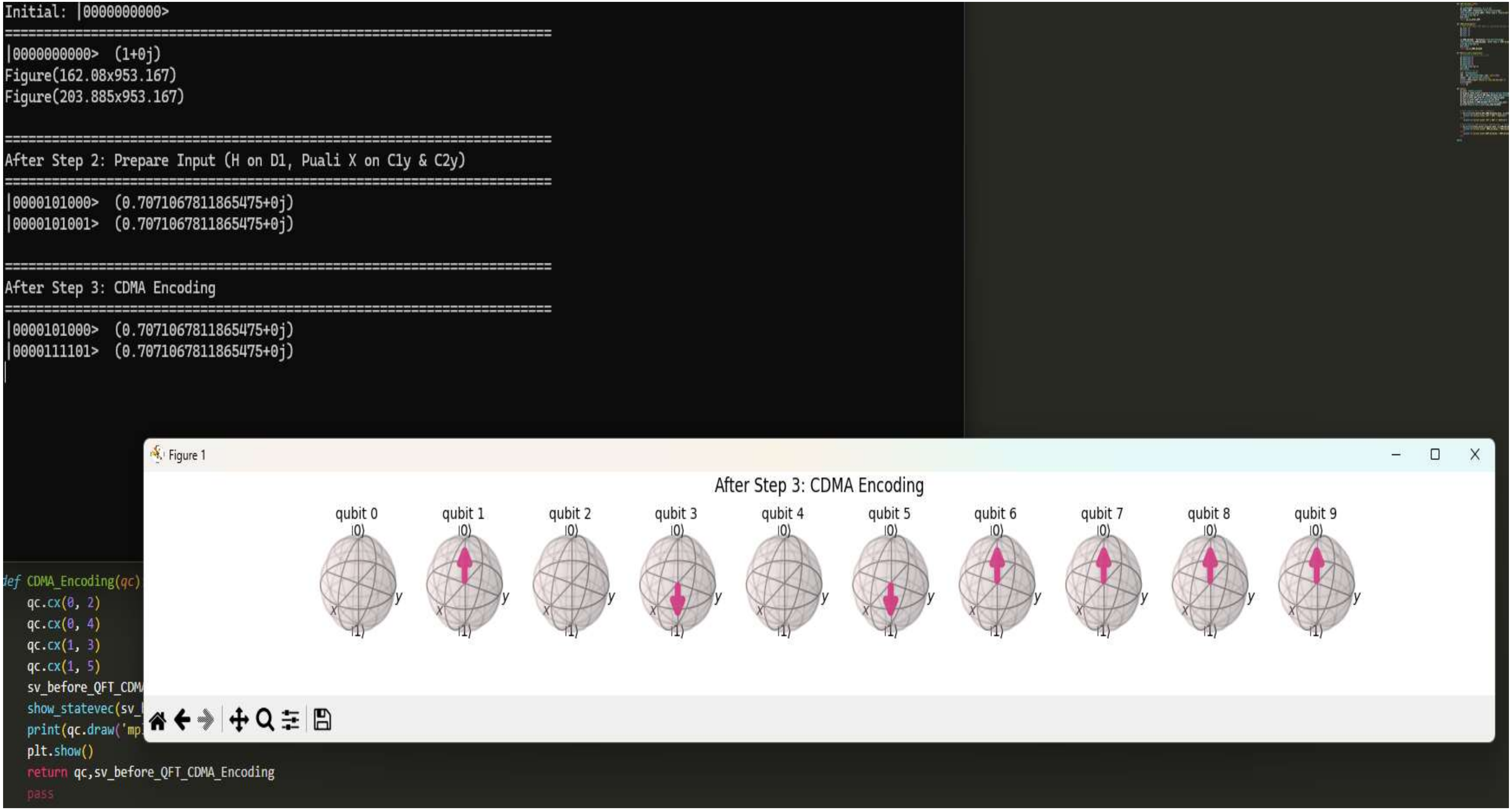
C:\Windows\system32\cmd.e

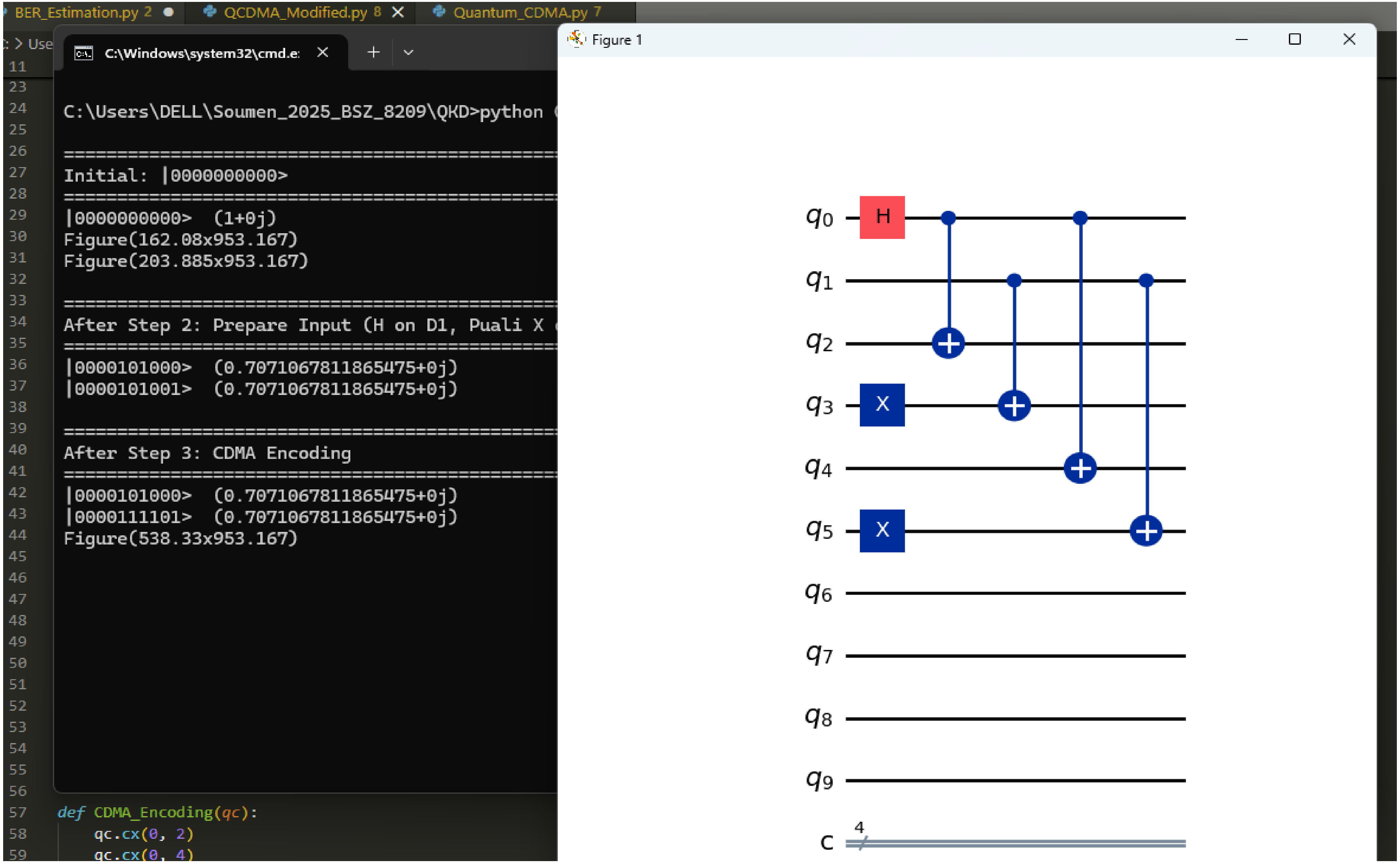
```
C:\Users\DELL\Soumen_2025_BSZ_8

==================================
Initial: |0000000000>
==================================
|0000000000>  (1+0j)
Figure(162.08x953.167)
Figure(203.885x953.167)
```

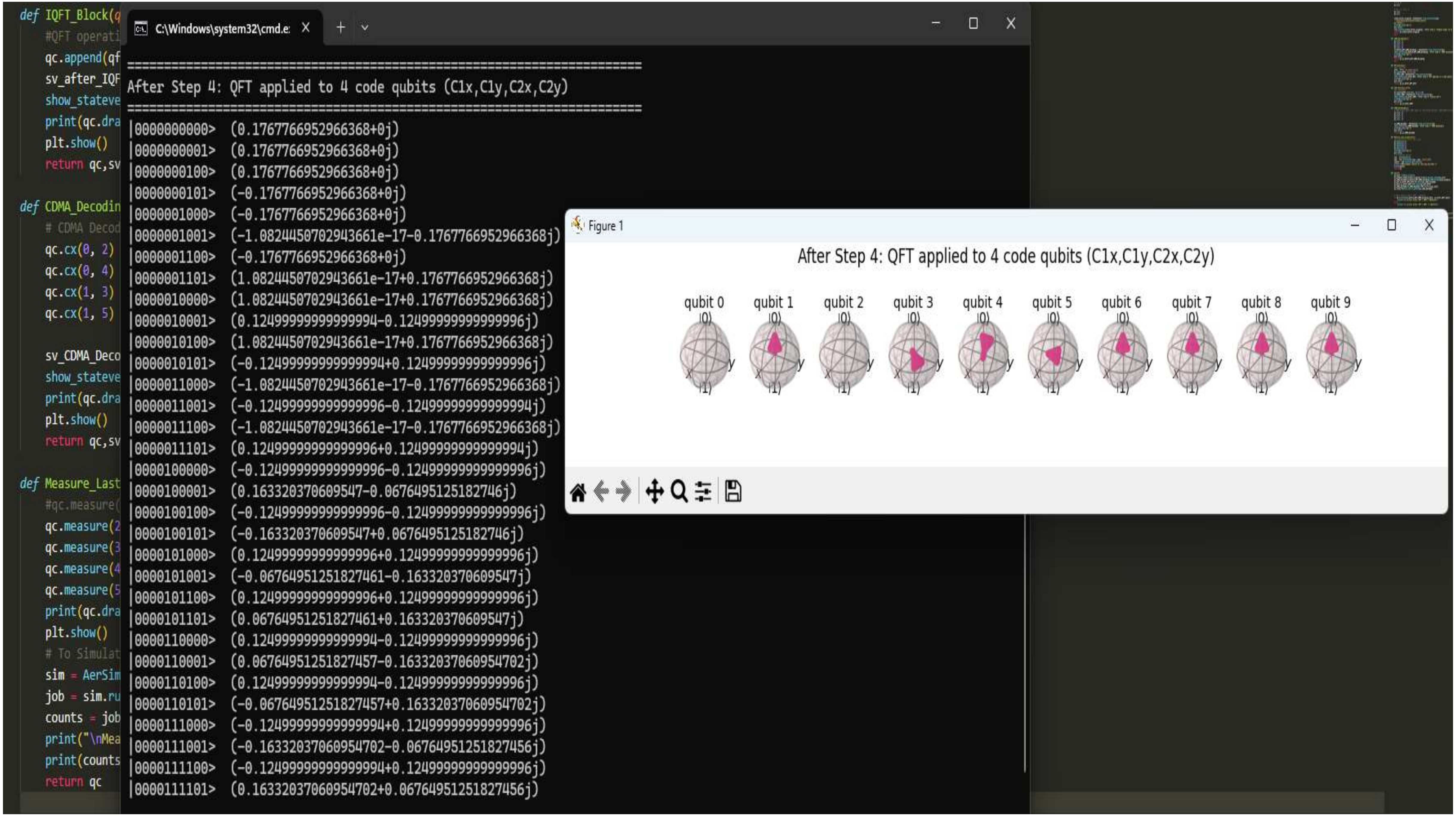
Figure 1




qc = QuantumCircuit(qr, cr)
# Simulate initial state
state_vector= Statevector.from_
'''plot_bloch_multivector(state
plt.show()'''
show_statevec(state_vector, "In
print(qc.draw('mpl'))
plt.show()
return qc
def Prepare_Initial_State(qc):
# H on D1
qc.h(0)
# C1y = 1, C2y = 1
qc.x(3)
qc.x(5)
C:\Windows\system32\cmd.e
C:\Users\DELL\Soumen_2025_BSZ_8209\QKD>python QCDMA_Modified.py
Initial: |0000000000>
|0000000000> (1+0j)
Figure(162.08x953.167)
Figure(203.885x953.167)
After Step 2: Prepare Input (H on D1, Puali X on C1y & C2y)
|0000101000> (0.7071067811865475+0j)
|0000101001> (0.7071067811865475+0j)
Figure 1
After Step 2: Prepare Input (H on D1, Puali X on C1y & C2y)
qubit 0
qubit 1
qubit 2
qubit 3
qubit 4
qubit 5
qubit 6
qubit 7
qubit 8
qubit 9


Initial: |0000000000>
|0000000000> (1+0j)
Figure(162.08x953.167)
Figure(203.885x953.167)
After Step 2: Prepare Input (H on D1, Puali X on C1y & C2y)
|0000101000> (0.7071067811865475+0j)
|0000101001> (0.7071067811865475+0j)
After Step 3: CDMA Encoding
|0000101000> (0.7071067811865475+0j)
|0000111101> (0.7071067811865475+0j)
def CDMA_Encoding(qc)
qc.cx(0, 2)
qc.cx(0, 4)
qc.cx(1, 3)
qc.cx(1, 5)
return qc,sv_before_QFT_CDMA_Encoding
pass
Figure 1
After Step 3: CDMA Encoding
qubit 0
qubit 1
qubit 2
qubit 3
qubit 4
qubit 5
qubit 6
qubit 7
qubit 8
qubit 9
|0⟩
|1⟩
x
y


BER_Estimation.py 2
QCDMA_Modified.py 8
Quantum_CDMA.py 7
C:\Windows\system32\cmd.e:
C:\Users\DELL\Soumen_2025_BSZ_8209\QKD>python
Initial: |0000000000>
|0000000000> (1+0j)
Figure(162.08x953.167)
Figure(203.885x953.167)
After Step 2: Prepare Input (H on D1, Puali X
|0000101000> (0.7071067811865475+0j)
|0000101001> (0.7071067811865475+0j)
After Step 3: CDMA Encoding
|0000101000> (0.7071067811865475+0j)
|0000111101> (0.7071067811865475+0j)
Figure(538.33x953.167)
def CDMA_Encoding(qc):
qc.cx(0, 2)
qc.cx(0, 4)
Figure 1
H
X
X
q0
q1
q2
q3
q4
q5
q6
q7
q8
q9
c
4

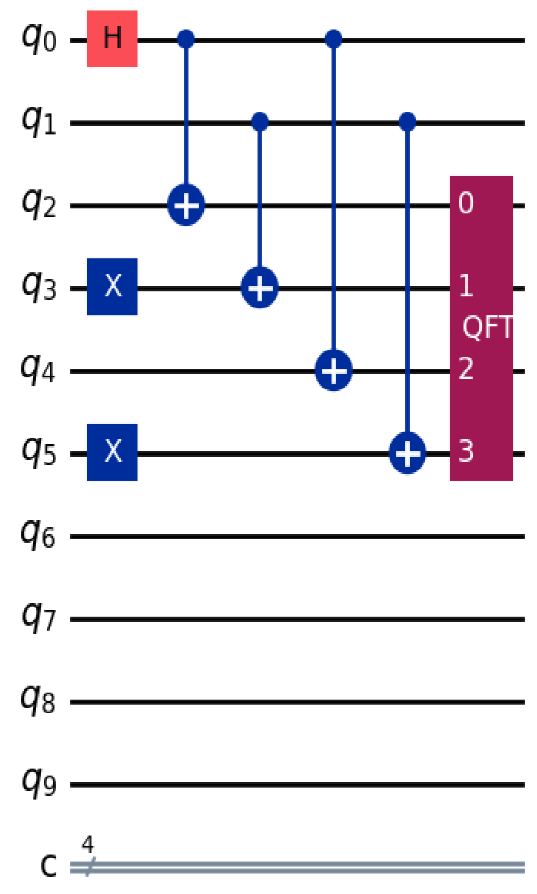

def IQFT_Block(q
#QFT operati
qc.append(qf
sv_after_IQF
show_stateve
print(qc.dra
plt.show()
return qc,sv
def CDMA_Decodin
# CDMA Decod
qc.cx(0, 2)
qc.cx(0, 4)
qc.cx(1, 3)
qc.cx(1, 5)
sv_CDMA_Deco
show_stateve
print(qc.dra
plt.show()
return qc,sv
def Measure_Last
#qc.measure(
qc.measure(2
qc.measure(3
qc.measure(4
qc.measure(5
print(qc.dra
plt.show()
# To Simulat
sim = AerSim
job = sim.ru
counts = job
print("\nMea
print(counts
return qc
C:\Windows\system32\cmd.e:
After Step 4: QFT applied to 4 code qubits (C1x,C1y,C2x,C2y)
|0000000000> (0.1767766952966368+0j)
|0000000001> (0.1767766952966368+0j)
|0000000100> (0.1767766952966368+0j)
|0000000101> (-0.1767766952966368+0j)
|0000001000> (-0.1767766952966368+0j)
|0000001001> (-1.0824450702943661e-17-0.1767766952966368j)
|0000001100> (-0.1767766952966368+0j)
|0000001101> (1.0824450702943661e-17+0.1767766952966368j)
|0000010000> (1.0824450702943661e-17+0.1767766952966368j)
|0000010001> (0.12499999999999994-0.12499999999999996j)
|0000010100> (1.0824450702943661e-17+0.1767766952966368j)
|0000010101> (-0.12499999999999994+0.12499999999999996j)
|0000011000> (-1.0824450702943661e-17-0.1767766952966368j)
|0000011001> (-0.12499999999999996-0.12499999999999994j)
|0000011100> (-1.0824450702943661e-17-0.1767766952966368j)
|0000011101> (0.12499999999999996+0.12499999999999994j)
|0000100000> (-0.12499999999999996-0.12499999999999996j)
|0000100001> (0.163320370609547-0.06764951251182746j)
|0000100100> (-0.12499999999999996-0.12499999999999996j)
|0000100101> (-0.163320370609547+0.06764951251182746j)
|0000101000> (0.12499999999999996+0.12499999999999996j)
|0000101001> (-0.06764951251827461-0.163320370609547j)
|0000101100> (0.12499999999999996+0.12499999999999996j)
|0000101101> (0.06764951251827461+0.163320370609547j)
|0000110000> (0.12499999999999994-0.12499999999999996j)
|0000110001> (0.06764951251827457-0.16332037060954702j)
|0000110100> (0.12499999999999994-0.12499999999999996j)
|0000110101> (-0.06764951251827457+0.16332037060954702j)
|0000111000> (-0.12499999999999994+0.12499999999999996j)
|0000111001> (-0.16332037060954702-0.06764951251827456j)
|0000111100> (-0.12499999999999994+0.12499999999999996j)
|0000111101> (0.16332037060954702+0.06764951251827456j)
Figure 1
After Step 4: QFT applied to 4 code qubits (C1x,C1y,C2x,C2y)
qubit 0
qubit 1
qubit 2
qubit 3
qubit 4
qubit 5
qubit 6
qubit 7
qubit 8
qubit 9

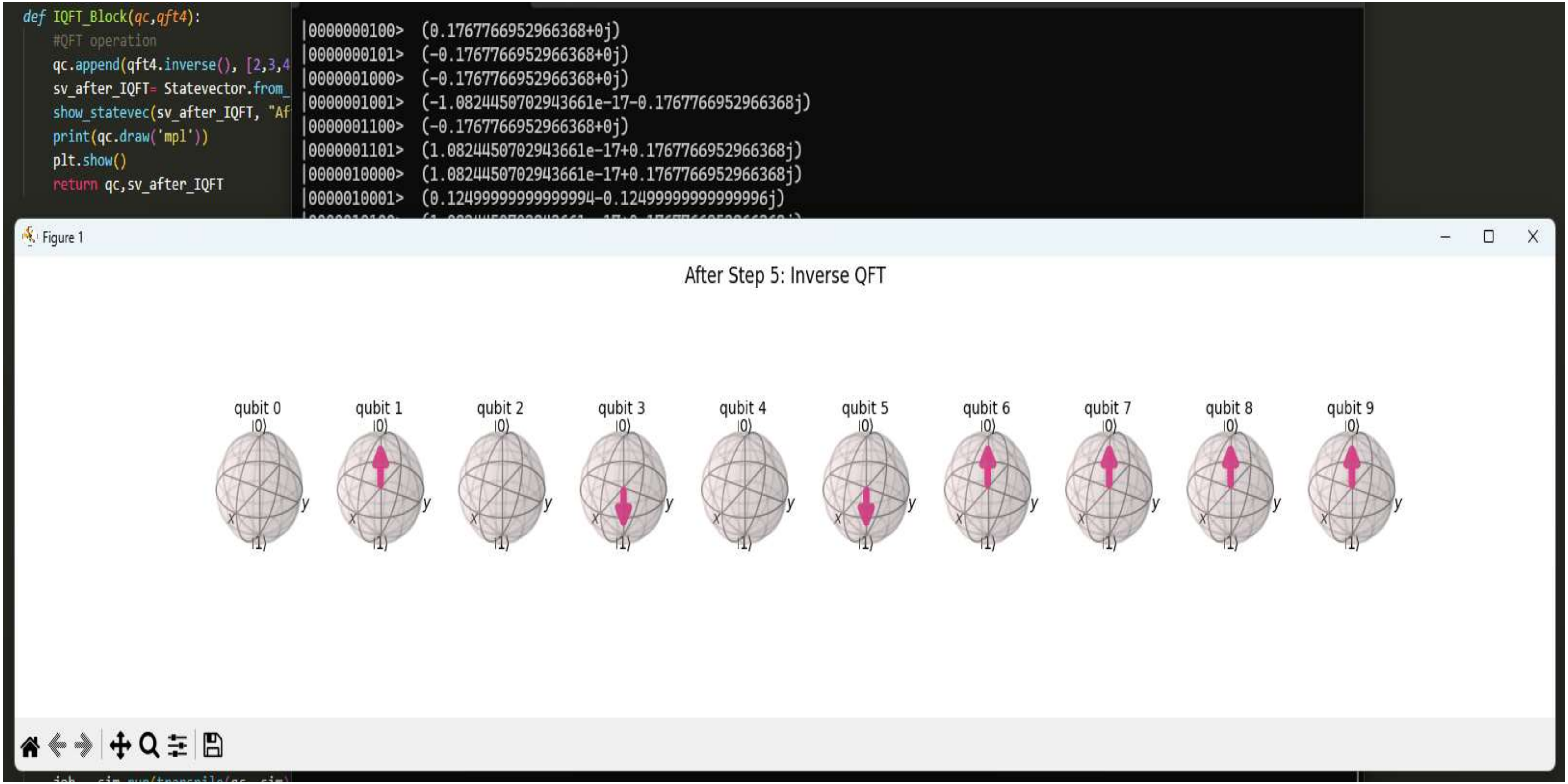
```
    plt.show()
    return qc,sv

def IQFT_Block(q
    #QFT operati
    qc.append(qf
    sv_after_IQF
    show_stateve
    print(qc.dra
    plt.show()
    return qc,sv

def CDMA_Decodin
    # CDMA Decod
    qc.cx(0, 2)
    qc.cx(0, 4)
    qc.cx(1, 3)
    qc.cx(1, 5)

    sv_CDMA_Deco
    show_stateve
    print(qc.dra
    plt.show()
    return qc,sv

def Measure_Last
    #qc.measure(
    qc.measure(2
    qc.measure(3
    qc.measure(4
    qc.measure(5
    print(qc.dra
    plt.show()
    # To Simulat
    sim = AerSim
    job = sim.ru
    counts = job
    print("\nMea
    print(counts
    return qc

def main():
```

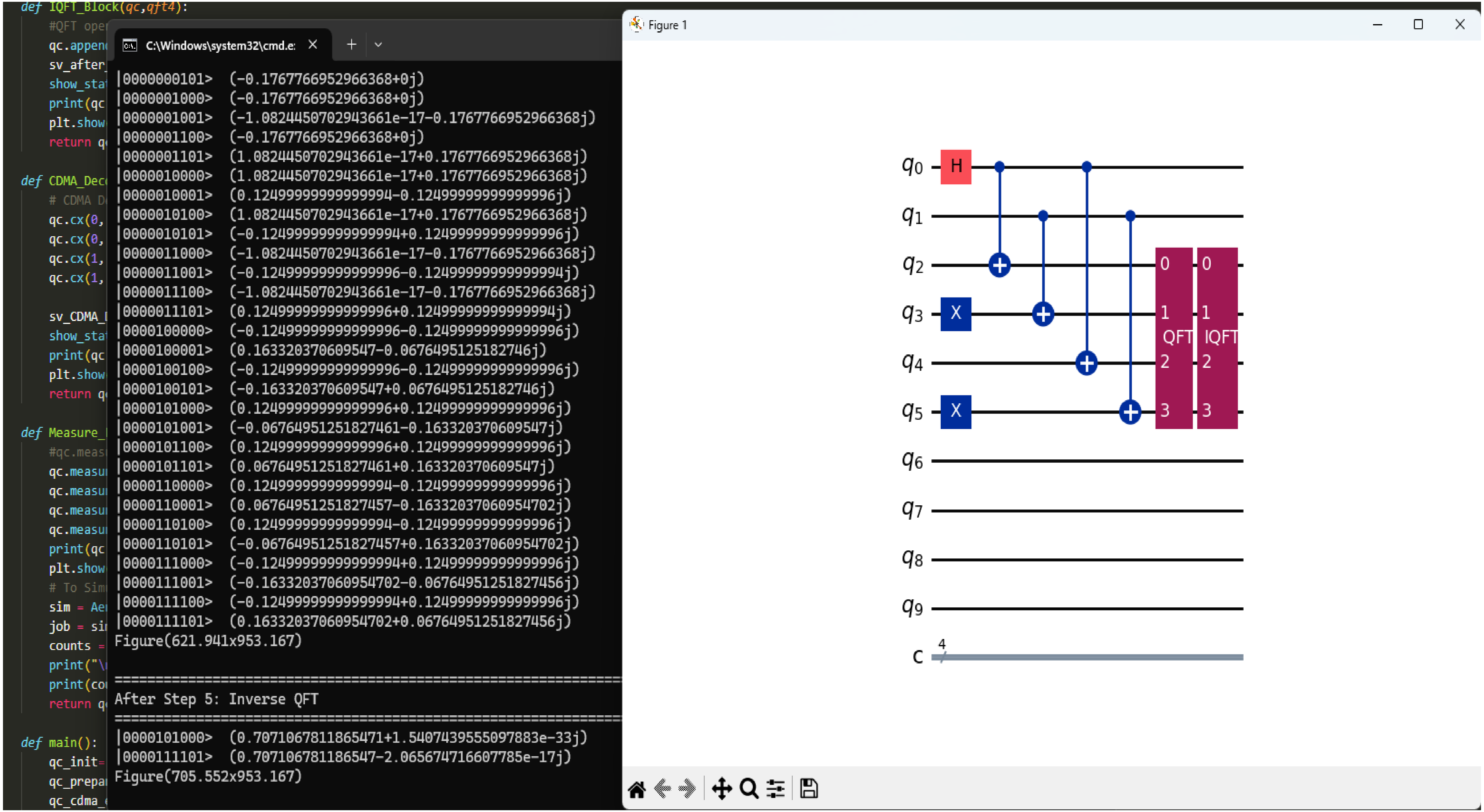
C:\Windows\system32\cmd.e

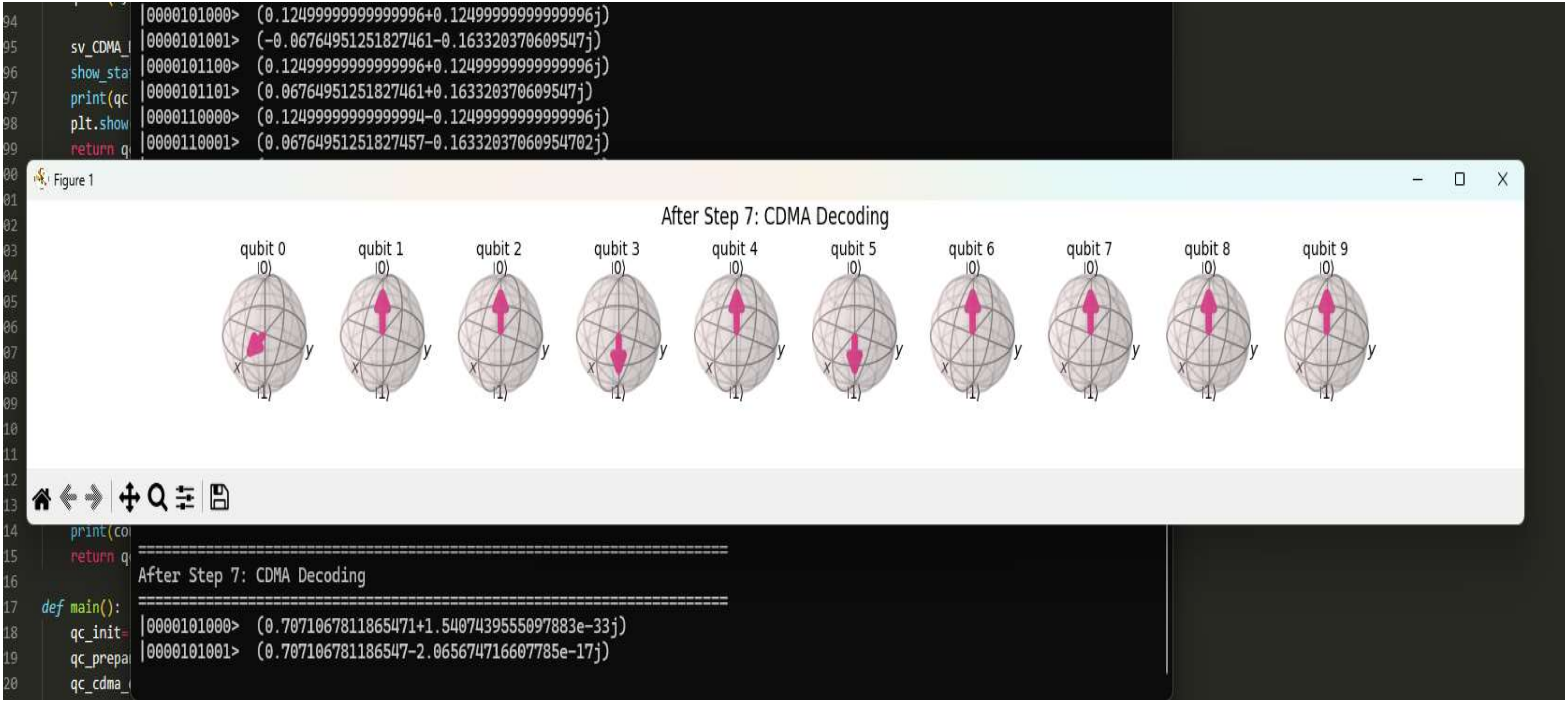
```
==========================================================
After Step 4: QFT applied to 4 code qubits (C1x,C1y,C2x,C2y)
==========================================================
|0000000000>  (0.1767766952966368+0j)
|0000000001>  (0.1767766952966368+0j)
|0000000100>  (0.1767766952966368+0j)
|0000000101>  (-0.1767766952966368+0j)
|0000001000>  (-0.1767766952966368+0j)
|0000001001>  (-1.0824450702943661e-17-0.1767766952966368j)
|0000001100>  (-0.1767766952966368+0j)
|0000001101>  (1.0824450702943661e-17+0.1767766952966368j)
|0000010000>  (1.0824450702943661e-17+0.1767766952966368j)
|0000010001>  (0.12499999999999994-0.12499999999999996j)
|0000010100>  (1.0824450702943661e-17+0.1767766952966368j)
|0000010101>  (-0.12499999999999994+0.12499999999999996j)
|0000011000>  (-1.0824450702943661e-17-0.1767766952966368j)
|0000011001>  (-0.12499999999999996-0.12499999999999994j)
|0000011100>  (-1.0824450702943661e-17-0.1767766952966368j)
|0000011101>  (0.12499999999999996+0.12499999999999994j)
|0000100000>  (-0.12499999999999996-0.12499999999999996j)
|0000100001>  (0.163320370609547-0.06764951251182746j)
|0000100100>  (-0.12499999999999996-0.12499999999999996j)
|0000100101>  (-0.163320370609547+0.06764951251182746j)
|0000101000>  (0.12499999999999996+0.12499999999999996j)
|0000101001>  (-0.06764951251827461-0.163320370609547j)
|0000101100>  (0.12499999999999996+0.12499999999999996j)
|0000101101>  (0.06764951251827461+0.163320370609547j)
|0000110000>  (0.12499999999999994-0.12499999999999996j)
|0000110001>  (0.06764951251827457-0.16332037060954702j)
|0000110100>  (0.12499999999999994-0.12499999999999996j)
|0000110101>  (-0.06764951251827457+0.16332037060954702j)
|0000111000>  (-0.12499999999999994+0.12499999999999996j)
|0000111001>  (-0.16332037060954702-0.06764951251827456j)
|0000111100>  (-0.12499999999999994+0.12499999999999996j)
|0000111101>  (0.16332037060954702+0.06764951251827456j)
Figure(621.941x953.167)
```

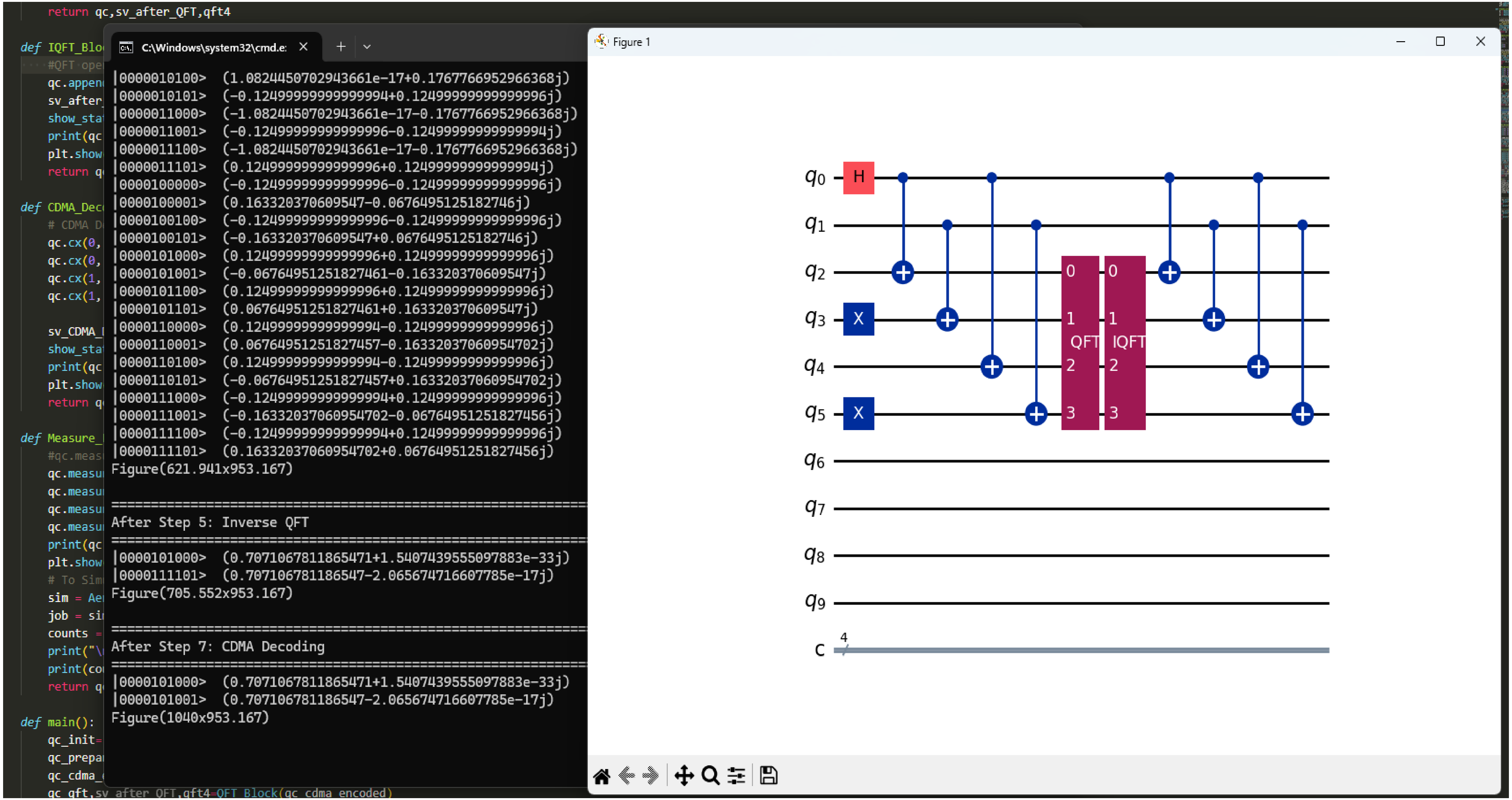
Figure 1

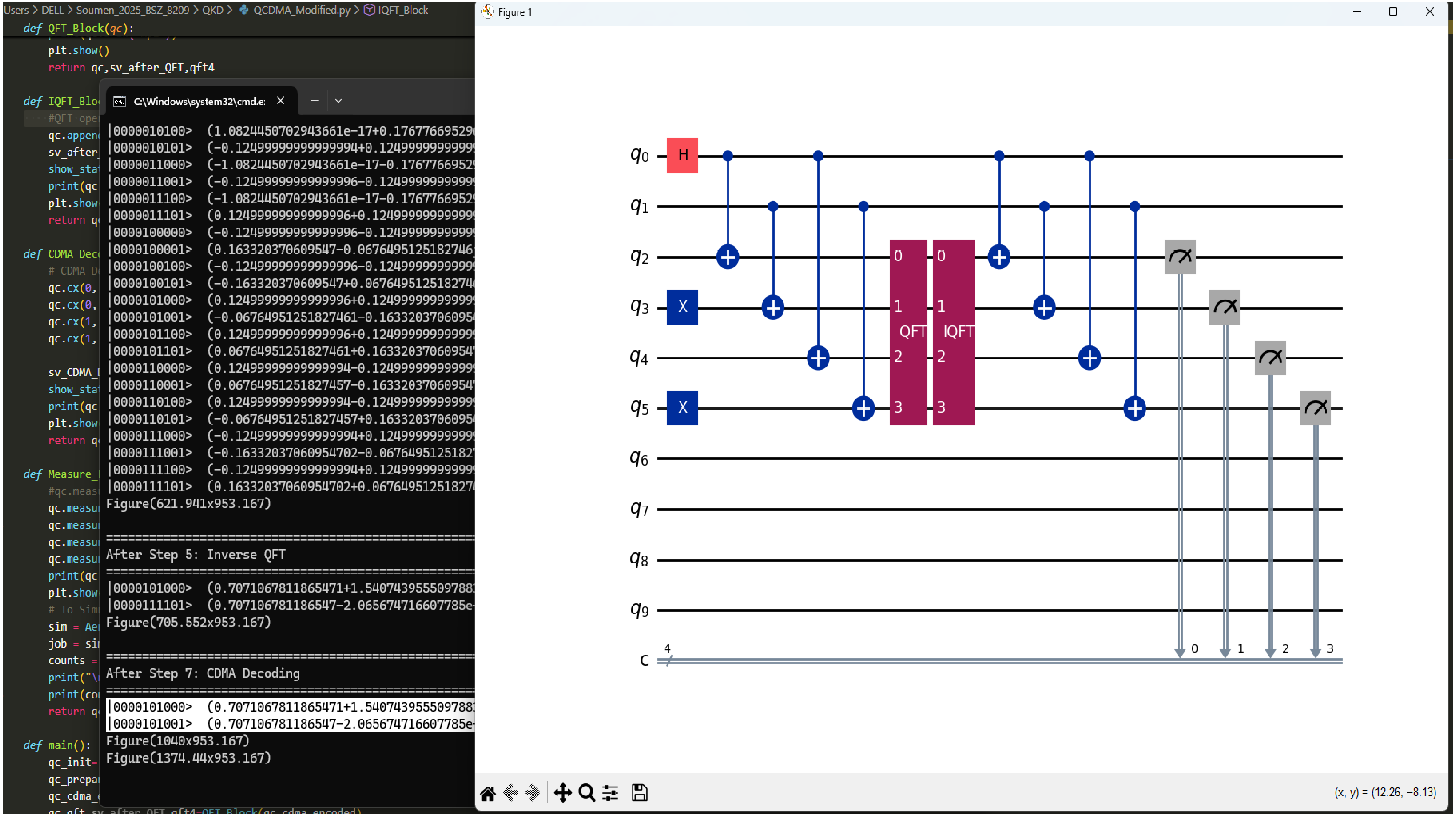



def IQFT_Block(qc,qft4):
#QFT operation
qc.append(qft4.inverse(), [2,3,4
sv_after_IQFT= Statevector.from_
show_statevec(sv_after_IQFT, "Af
print(qc.draw('mpl'))
plt.show()
return qc,sv_after_IQFT
|0000000100> (0.1767766952966368+0j)
|0000000101> (-0.1767766952966368+0j)
|0000001000> (-0.1767766952966368+0j)
|0000001001> (-1.0824450702943661e-17-0.1767766952966368j)
|0000001100> (-0.1767766952966368+0j)
|0000001101> (1.0824450702943661e-17+0.1767766952966368j)
|0000010000> (1.0824450702943661e-17+0.1767766952966368j)
|0000010001> (0.12499999999999994-0.12499999999999996j)
Figure 1
After Step 5: Inverse QFT
qubit 0
qubit 1
qubit 2
qubit 3
qubit 4
qubit 5
qubit 6
qubit 7
qubit 8
qubit 9

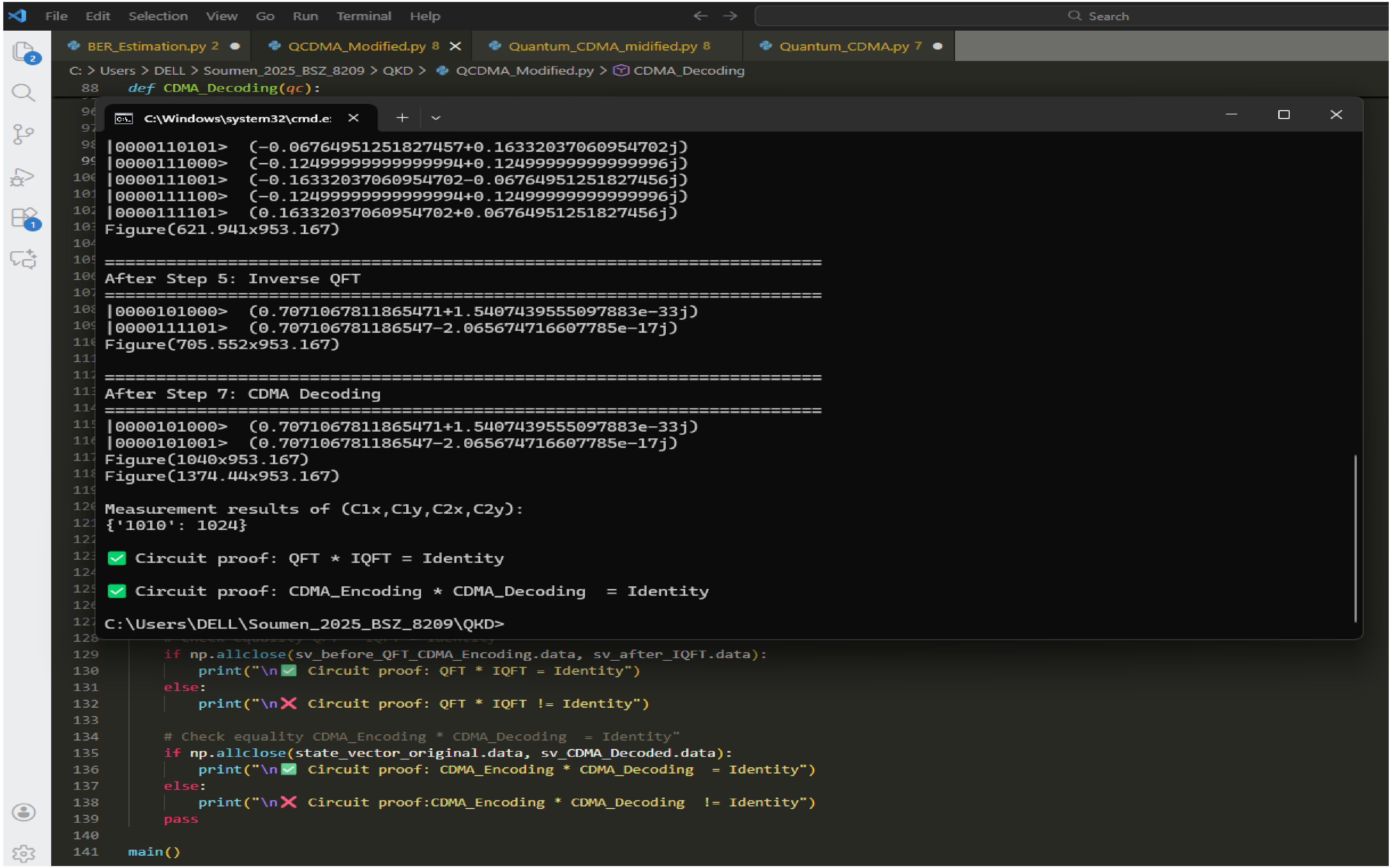

C:\Windows\system32\cmd.e:
|0000000101> (-0.1767766952966368+0j)
|0000001000> (-0.1767766952966368+0j)
|0000001001> (-1.0824450702943661e-17-0.1767766952966368j)
|0000001100> (-0.1767766952966368+0j)
|0000001101> (1.0824450702943661e-17+0.1767766952966368j)
|0000010000> (1.0824450702943661e-17+0.1767766952966368j)
|0000010001> (0.12499999999999994-0.12499999999999996j)
|0000010100> (1.0824450702943661e-17+0.1767766952966368j)
|0000010101> (-0.12499999999999994+0.12499999999999996j)
|0000011000> (-1.0824450702943661e-17-0.1767766952966368j)
|0000011001> (-0.12499999999999996-0.12499999999999994j)
|0000011100> (-1.0824450702943661e-17-0.1767766952966368j)
|0000011101> (0.12499999999999996+0.12499999999999994j)
|0000100000> (-0.12499999999999996-0.12499999999999996j)
|0000100001> (0.163320370609547-0.0676495125182746j)
|0000100100> (-0.12499999999999996-0.12499999999999996j)
|0000100101> (-0.163320370609547+0.0676495125182746j)
|0000101000> (0.12499999999999996+0.12499999999999996j)
|0000101001> (-0.06764951251827461-0.163320370609547j)
|0000101100> (0.12499999999999996+0.12499999999999996j)
|0000101101> (0.06764951251827461+0.163320370609547j)
|0000110000> (0.12499999999999994-0.12499999999999996j)
|0000110001> (0.06764951251827457-0.16332037060954702j)
|0000110100> (0.12499999999999994-0.12499999999999996j)
|0000110101> (-0.06764951251827457+0.16332037060954702j)
|0000111000> (-0.12499999999999994+0.12499999999999996j)
|0000111001> (-0.16332037060954702-0.06764951251827456j)
|0000111100> (-0.12499999999999994+0.12499999999999996j)
|0000111101> (0.16332037060954702+0.06764951251827456j)
Figure(621.941x953.167)
After Step 5: Inverse QFT
|0000101000> (0.7071067811865471+1.5407439555097883e-33j)
|0000111101> (0.707106781186547-2.065674716607785e-17j)
Figure(705.552x953.167)
Figure 1
q0
q1
q2
q3
q4
q5
q6
q7
q8
q9
c
4
H
X
X
QFT
IQFT
0
1
2
3


|0000101000> (0.12499999999999996+0.12499999999999996j)
|0000101001> (-0.06764951251827461-0.163320370609547j)
|0000101100> (0.12499999999999996+0.12499999999999996j)
|0000101101> (0.06764951251827461+0.163320370609547j)
|0000110000> (0.12499999999999994-0.12499999999999996j)
|0000110001> (0.06764951251827457-0.16332037060954702j)
Figure 1
After Step 7: CDMA Decoding
qubit 0
qubit 1
qubit 2
qubit 3
qubit 4
qubit 5
qubit 6
qubit 7
qubit 8
qubit 9
|0⟩
|1⟩
x
y
def main():
After Step 7: CDMA Decoding
|0000101000> (0.7071067811865471+1.5407439555097883e-33j)
|0000101001> (0.707106781186547-2.065674716607785e-17j)


return qc,sv_after_QFT,qft4
def IQFT_Blo
#QFT ope
qc.appen
sv_after
show_sta
print(qc
plt.show
return q
def CDMA_Dec
CDMA D
qc.cx(0,
qc.cx(0,
qc.cx(1,
qc.cx(1,
sv_CDMA_
show_sta
print(qc
plt.show
return q
def Measure_
#qc.meas
qc.measu
qc.measu
qc.measu
qc.measu
print(qc
plt.show
# To Sim
sim = Ae
job = si
counts =
print("\
print(co
return q
def main():
qc_init=
qc_prepa
qc_cdma_
qc_qft,sv_after_QFT,qft4=QFT_Block(qc_cdma_encoded)
C:\Windows\system32\cmd.e
|0000010100> (1.0824450702943661e-17+0.1767766952966368j)
|0000010101> (-0.12499999999999994+0.12499999999999996j)
|0000011000> (-1.0824450702943661e-17-0.1767766952966368j)
|0000011001> (-0.12499999999999996-0.12499999999999994j)
|0000011100> (-1.0824450702943661e-17-0.1767766952966368j)
|0000011101> (0.12499999999999996+0.12499999999999994j)
|0000100000> (-0.12499999999999996-0.12499999999999996j)
|0000100001> (0.16332037060954 7-0.0676495125182746j)
|0000100100> (-0.12499999999999996-0.12499999999999996j)
|0000100101> (-0.16332037060954 7+0.0676495125182746j)
|0000101000> (0.12499999999999996+0.12499999999999996j)
|0000101001> (-0.06764951251827461-0.163320370609547j)
|0000101100> (0.12499999999999996+0.12499999999999996j)
|0000101101> (0.06764951251827461+0.163320370609547j)
|0000110000> (0.12499999999999994-0.12499999999999996j)
|0000110001> (0.06764951251827457-0.16332037060954702j)
|0000110100> (0.12499999999999994-0.12499999999999996j)
|0000110101> (-0.06764951251827457+0.16332037060954702j)
|0000111000> (-0.12499999999999994+0.12499999999999996j)
|0000111001> (-0.16332037060954702-0.06764951251827456j)
|0000111100> (-0.12499999999999994+0.12499999999999996j)
|0000111101> (0.16332037060954702+0.06764951251827456j)
Figure(621.941x953.167)
After Step 5: Inverse QFT
|0000101000> (0.7071067811865471+1.5407439555097883e-33j)
|0000111101> (0.707106781186547-2.065674716607785e-17j)
Figure(705.552x953.167)
After Step 7: CDMA Decoding
|0000101000> (0.7071067811865471+1.5407439555097883e-33j)
|0000101001> (0.707106781186547-2.065674716607785e-17j)
Figure(1040x953.167)
Figure 1
q_0
q_1
q_2
q_3
q_4
q_5
q_6
q_7
q_8
q_9
c
4
H
X
X
QFT
IQFT
0
1
2
3



Users > DELL > Soumen_2025_BSZ_8209 > QKD > QCDMA_Modified.py > IQFT_Block
def QFT_Block(qc):
plt.show()
return qc,sv_after_QFT,qft4
def IQFT_Blo
#QFT ope
qc.append
sv_after
show_sta
print(qc
plt.show
return q
def CDMA_Dec
CDMA D
qc.cx(0,
qc.cx(0,
qc.cx(1,
qc.cx(1,
sv_CDMA_
show_sta
print(qc
plt.show
return q
def Measure_
#qc.meas
qc.measu
qc.measu
qc.measu
qc.measu
print(qc
plt.show
# To Sim
sim = Ae
job = si
counts =
print("\
print(co
return q
def main():
qc_init=
qc_prepa
qc_cdma_
C:\Windows\system32\cmd.e
|0000010100> (1.0824450702943661e-17+0.17677669529
|0000010101> (-0.12499999999999994+0.124999999999999
|0000011000> (-1.0824450702943661e-17-0.176776695
|0000011001> (-0.12499999999999996-0.124999999999999
|0000011100> (-1.0824450702943661e-17-0.176776695
|0000011101> (0.12499999999999996+0.124999999999999
|0000100000> (-0.12499999999999996-0.124999999999999
|0000100001> (0.163320370609547-0.0676495125182746
|0000100100> (-0.12499999999999996-0.124999999999999
|0000100101> (-0.163320370609547+0.0676495125182746
|0000101000> (0.12499999999999996+0.124999999999999
|0000101001> (-0.06764951251827461-0.1633203706095
|0000101100> (0.12499999999999996+0.124999999999999
|0000101101> (0.06764951251827461+0.163320370609547
|0000110000> (0.12499999999999994-0.124999999999999
|0000110001> (0.06764951251827457-0.163320370609547
|0000110100> (0.12499999999999994-0.124999999999999
|0000110101> (-0.06764951251827457+0.1633203706095
|0000111000> (-0.12499999999999994+0.124999999999999
|0000111001> (-0.16332037060954702-0.0676495125182
|0000111100> (-0.12499999999999994+0.124999999999999
|0000111101> (0.16332037060954702+0.0676495125182
Figure(621.941x953.167)
After Step 5: Inverse QFT
|0000101000> (0.7071067811865471+1.540743955509788
|0000111101> (0.707106781186547-2.065674716607785e-
Figure(705.552x953.167)
After Step 7: CDMA Decoding
|0000101000> (0.7071067811865471+1.540743955509788
|0000101001> (0.707106781186547-2.065674716607785e-
Figure(1040x953.167)
Figure(1374.44x953.167)
Figure 1
q0
q1
q2
q3
q4
q5
q6
q7
q8
q9
c
H
X
X
0
1
2
3
QFT
0
1
2
3
IQFT
4
0
1
2
3
(x, y) = (12.26, −8.13)


File Edit Selection View Go Run Terminal Help
Search
BER_Estimation.py 2
QCDMA_Modified.py 8
Quantum_CDMA_midified.py 8
Quantum_CDMA.py 7
C: > Users > DELL > Soumen_2025_BSZ_8209 > QKD > QCDMA_Modified.py > CDMA_Decoding
def CDMA_Decoding(qc):
C:\Windows\system32\cmd.e:
|0000110101> (-0.06764951251827457+0.16332037060954702j)
|0000111000> (-0.12499999999999994+0.12499999999999996j)
|0000111001> (-0.16332037060954702-0.06764951251827456j)
|0000111100> (-0.12499999999999994+0.12499999999999996j)
|0000111101> (0.16332037060954702+0.06764951251827456j)
Figure(621.941x953.167)
After Step 5: Inverse QFT
|0000101000> (0.7071067811865471+1.5407439555097883e-33j)
|0000111101> (0.707106781186547-2.065674716607785e-17j)
Figure(705.552x953.167)
After Step 7: CDMA Decoding
|0000101000> (0.7071067811865471+1.5407439555097883e-33j)
|0000101001> (0.707106781186547-2.065674716607785e-17j)
Figure(1040x953.167)
Figure(1374.44x953.167)
Measurement results of (C1x,C1y,C2x,C2y):
{'1010': 1024}
Circuit proof: QFT * IQFT = Identity
Circuit proof: CDMA_Encoding * CDMA_Decoding = Identity
C:\Users\DELL\Soumen_2025_BSZ_8209\QKD>
if np.allclose(sv_before_QFT_CDMA_Encoding.data, sv_after_IQFT.data):
print("\n Circuit proof: QFT * IQFT = Identity")
else:
print("\n Circuit proof: QFT * IQFT != Identity")
# Check equality CDMA_Encoding * CDMA_Decoding = Identity"
if np.allclose(state_vector_original.data, sv_CDMA_Decoded.data):
print("\n Circuit proof: CDMA_Encoding * CDMA_Decoding = Identity")
else:
print("\n Circuit proof:CDMA_Encoding * CDMA_Decoding != Identity")
pass
main()

- Using QFT we transforms the Qubit from one-base to another to increases security by spreading ,each user information across multiple new basis vectors.

- QFT ,scaled linearly from N to N+1 Qubits order whereas classical DFT can scale on powers of 2.

- But as the number of Qubits increased the QFT operation is going to exponentially complex , Can we have a polynomial time solution in near future ?

- Can we also able to reduce the circuit complexity without the loss of generality. Hence the Optimization.

References

1. Nielsen, M.A., Chuang, I.L.: Quantum Computation and Quantum Information, 2nd edn. Cambridge University Press/Massachusetts Institute of Technology, Cam bridge (2000)
2. Science Direct. https://www.sciencedirect.com/topics/computer-science/ quantum-circuit. Accessed 9 Oct 2020
3. Sharma, V., Banerjee, S.: Quantum communication using code division multiple access network. Opt. Quant. Electron. 52, 381 (2020). https://doi.org/10.1007/ s11082-020-02494-3
4. Tan, X., Cheng, S., Li, J., Feng, Z.: Quantum key distribution protocol using quan tum Fourier transform. In: 2015 IEEE 29th International Conference on Advanced Information Networking and Applications Workshops, Gwangiu, pp. 96–101 (2015). https://doi.org/10.1109/WAINA.2015.8
5. Kumavor, P.D., Beal, A.C., Yelin, S., Donkor, E., Wang, B.C.: Comparison of four multi-user quantum key distribution schemes over passive optical networks. J. Lightwave Technol. 23(1), 268–276 (2005). https://doi.org/10.1109/JLT.2004. 834481
6. Brassard, G., Bussieres, F., Godbout, N., Lacroix, S.: Multiuser quantum key dis tribution using wavelength division multiplexing. In: Proceedings of SPIE 5260, Applications of Photonic Technology 6, (2003). https://doi.org/10.1117/12.543338
7. Xue, P., Wang, K., Wang, X.: Efficient multiuser quantum cryptography net work based on entanglement. Sci. Rep. 7, 45928 (2017). https://doi.org/10.1038/ srep45928
8. 10. Quirk- A drag-and-drop quantum circuit simulator. https://algassert.com/quirk. Accessed 9 Oct 2020 11. qcircuit-Macros to generate quantum circuits. https://ctan.org/pkg/qcircuit. Accessed 9 Oct 2020

Thank you