Furious Five: QuEra Technical Challenge

Chris Liu, Ava Brule, Ryan Ruiz, Claire Mao, Krisztina Tolotti

This document contains a detailed explanation of our approach to the QuEra Technical Challenge as well as all the code involved in our final submission.

Background

We started by reading the attached resources in the challenge description. As a freshman-only team, we struggled to understand many technical details related to the challenge. Instead of giving up, we took a heuristic approach to the technique and started by implementing the two quantum circuits depicted in the document.

We mapped out the individual gates ($\sqrt{Y}^\dagger$, $\sqrt{Y}$, CZ, $CNOT$) to the appropriately numbered qubits and initially implemented the circuit directly in `stim`:

```
SQRT_Y_DAG 0 1 2 3 4 5
CZ 1 2 3 4 5 6
SQRT_Y 6
CZ 0 3 2 5 4 6
SQRT_Y 2 3 4 5 6
CZ 0 1 2 3 4 5
SQRT_Y 1 2 4

H 13
SQRT_Y_DAG 7 8 9 10 11 12
CZ 8 9 10 11 12 13
SQRT_Y 13
CZ 7 10 9 12 11 13
SQRT_Y 9 10 11 12 13
CZ 7 8 9 10 11 12
SQRT_Y 8 9 11

CX 0 7 1 8 2 9 3 10 4 11 5 12 6 13
MZ 7 8 9 10 11 12 13

<more gates>

CZ rec[-7] 0 rec[-6] 1 rec[-5] 2 rec[-4] 3 rec[-3] 4 rec[-2] 5 rec[-1] 6
CX rec[-14] 0 rec[-13] 1 rec[-12] 2 rec[-11] 3 rec[-10] 4 rec[-9] 5 rec[-8] 6

SQRT_Y_DAG 1 2 4
CZ 0 1 2 3 4 5
SQRT_Y_DAG 2 3 4 5 6
CZ 0 3 2 5 4 6
SQRT_Y_DAG 6
CZ 1 2 3 4 5 6
SQRT_Y 0 1 2 3 4 5

MZ 0 1 2 3 4 5 6
```

Due to an inaccurate understanding of how the error correction code works, we attempted to undo the MSD encoding circuit at the end by applying its inverse. During lunch time, we asked for help from the QuEra staff. They were kind and explained to us the details of the error correction mechanism as well as how the referenced libraries are intended to be used (`stim` for Clifford-only simulation and `tsim` for T-gates).

For the remainder of the day, we synthesized our new understanding of the problem as well as various heuristic engineering techniques into the solution described in this Jupyter Notebook.

The solution

Our solution to the challenge is exploratory in nature. We will explain each individual part as we demonstrate the corresponding code.

In [1]:
```
# Part 0: Setup

import numpy as np
import bloqade
import bloqade.stim
import bloqade.tsim
from bloqade import squin
from kirin.dialects.ilist import IList
```

Part 1: MSD encode

We implemented the MSD encode circuit in `squin` and we simulated it using `stim`. We considered cases where there is no noise as well as a constant depolarizing noise at the end of the circuit. We calculated the syndromes as well as the logical observable after measuring the qubits. We noticed that while the physical observable is consistent with the input state $|0\rangle$, two of the syndromes are consistent of the opposite polarity (-1 instead of 1). While we were unable to understand why this is the case, we applied an X gate to the third qubit which flips all three syndromes back to 1.

We sampled the circuit and dropped shots where the syndromes indicate an error. We calculated the corresponding logical and physical errors and noticed a significant decrease in error rate after the syndrome-filtering mechanism.

MSD circuit:

```
q0: -SQRT_Y_DAG---@---------------@-------------DEPOLARIZE1(0.2)-M(0):rec[0]------------------
                  |               |
q1: -SQRT_Y_DAG-@-|---------------@------SQRT_Y-DEPOLARIZE1(0.2)-M(0):rec[1]------------------
                | |
q2: -SQRT_Y_DAG-@-|------@-SQRT_Y-@------SQRT_Y-DEPOLARIZE1(0.2)-M(0):rec[2]------------------
                  |      |        |
q3: -SQRT_Y_DAG-@-@------|-SQRT_Y-@-------------DEPOLARIZE1(0.2)-X----------------M(0):rec[3]-
                |        |
q4: -SQRT_Y_DAG-@--------|-@------SQRT_Y-@------SQRT_Y-----------DEPOLARIZE1(0.2)-M(0):rec[4]-
                         | |             |
q5: -SQRT_Y_DAG-@--------@-|------SQRT_Y-@-----------------------DEPOLARIZE1(0.2)-M(0):rec[5]-
                |          |
q6: ------------@-SQRT_Y---@------SQRT_Y-------------------------DEPOLARIZE1(0.2)-M(0):rec[6]-
```

In [ ]:
```
# Part 1: MSD encode


@squin.kernel
def msd_0():
    """Equivalent stim code:

    SQRT_Y_DAG 0 1 2 3 4 5
    CZ 1 2 3 4 5 6
    SQRT_Y 6
    CZ 0 3 2 5 4 6
    SQRT_Y 2 3 4 5 6
    CZ 0 1 2 3 4 5
    SQRT_Y 1 2 4
    """

    q = squin.qalloc(7)

    squin.broadcast.sqrt_y_adj(q[:6])
    squin.broadcast.cz(IList([q[1], q[3], q[5]]), IList([q[2], q[4], q[6]]))
    squin.sqrt_y(q[6])
    squin.broadcast.cz(IList([q[0], q[2], q[4]]), IList([q[3], q[5], q[6]]))
    squin.broadcast.sqrt_y(q[2:7])
```

```python
    squin.broadcast.cz(IList([q[0], q[2], q[4]]), IList([q[1], q[3], q[5]]))
    squin.broadcast.sqrt_y(IList([q[1], q[2], q[4]]))

    # optional noise
    squin.broadcast.depolarize(0.2, q)

    # flipping qubit 3 because some syndromes are -1???
    squin.x(q[3])
    squin.broadcast.measure(q)


circ = bloqade.stim.Circuit(msd_0)
circ.diagram()
```

In [3]:
```python
def sample(circ, shots=100):
    """Sample from the circuit and convert to -1/+1"""
    return circ.compile_sampler().sample(shots=shots) * (-2) + 1


def logical_obs(arr):
    """Calculate logical observables from measurement results"""
    return arr[:, 0] * arr[:, 1] * arr[:, 5]


def syndromes(arr):
    """Calculate syndromes from measurement results"""
    return [
        arr[:, 0] * arr[:, 1] * arr[:, 2] * arr[:, 3],
        arr[:, 1] * arr[:, 5] * arr[:, 4] * arr[:, 2],
        arr[:, 6] * arr[:, 4] * arr[:, 2] * arr[:, 3],
    ]


def noise_free_mask(arr):
    """Get mask of shots with no noise (all syndromes +1)"""
    syn = syndromes(arr)
    return syn[0] + syn[1] + syn[2] == 3


def logical_error(values, true_value=1):
    """Calculate logical error rate given logical values and true value"""
    if len(values) == 0:
        return float("nan")
    return float(((values != true_value).sum()) / len(values))


samples = sample(circ)
values = logical_obs(samples)
synds = syndromes(samples)
mask = noise_free_mask(samples)

print("Values: ", values)
print("Syndromes: ", synds)
print("Noise-free mask: ", mask)
print("Values (no noise): ", values[mask])
print("Logical error rate: ", logical_error(values[mask]))
```

```
Values:  [-1 -1  1 -1  1  1 -1  1 -1  1  1  1  1  1 -1  1  1 -1 -1  1  1 -1  1  1
 -1 -1  1  1  1  1  1  1 -1 -1  1  1  1  1  1  1  1  1  1 -1  1  1 -1  1
  1  1  1  1  1  1 -1 -1  1  1 -1 -1 -1 -1 -1  1 -1  1  1  1  1  1  1  1
  1 -1 -1 -1 -1 -1  1 -1  1 -1  1 -1  1  1  1  1 -1  1  1 -1 -1  1 -1  1
  1  1  1  1]
Syndromes:  [array([ 1, -1,  1,  1,  1,  1, -1, -1,  1, -1,  1, -1,  1,  1,  1,  1,  1,
        1, -1, -1,  1,  1, -1, -1,  1, -1,  1, -1,  1,  1,  1, -1, -1,  1,
        1,  1,  1, -1,  1,  1,  1,  1, -1, -1,  1,  1, -1,  1, -1,  1, -1,
       -1, -1,  1, -1,  1, -1,  1, -1, -1, -1, -1, -1,  1,  1,  1,  1, -1,
        1,  1,  1,  1, -1, -1, -1,  1, -1,  1,  1, -1,  1,  1,  1, -1,  1,
        1, -1,  1, -1,  1,  1, -1,  1,  1, -1,  1, -1, -1,  1,  1]), array([-1, -1, -1, -1, -1, -1,  1,  1, -1,
1,  1,  1,  1,  1, -1,  1,  1,
       -1,  1,  1,  1, -1, -1, -1, -1, -1,  1,  1,  1, -1,  1, -1,  1,  1,
        1,  1,  1,  1,  1,  1,  1, -1, -1,  1,  1,  1,  1,  1,  1, -1,  1,
        1, -1,  1,  1, -1, -1,  1,  1,  1,  1,  1,  1,  1, -1,  1,  1, -1,
        1,  1,  1,  1,  1, -1, -1, -1,  1, -1, -1, -1,  1,  1,  1, -1,  1,
       -1, -1,  1,  1,  1,  1,  1, -1,  1, -1,  1, -1,  1,  1, -1]), array([-1, -1, -1, -1, -1, -1,  1, -1, -1, -
1,  1, -1,  1,  1, -1, -1,  1,
       -1,  1,  1,  1,  1,  1,  1,  1, -1, -1, -1,  1,  1,  1, -1,  1,  1,
        1,  1, -1, -1,  1,  1,  1,  1, -1,  1,  1,  1,  1,  1, -1, -1, -1,
       -1,  1,  1, -1,  1, -1,  1, -1,  1,  1, -1,  1,  1,  1,  1,  1, -1,
        1,  1,  1,  1,  1,  1,  1,  1,  1,  1, -1,  1, -1, -1,  1,  1,  1,
       -1,  1, -1,  1, -1,  1, -1,  1,  1,  1,  1, -1, -1,  1, -1])]
Noise-free mask:  [False False False False False False False False False False  True False
  True  True False False  True False False False  True False False False
 False False False False  True False  True False False  True  True  True
 False False  True  True  True False False False  True  True False  True
 False False False False False  True False False False  True False False
 False False False  True False  True  True False  True  True  True  True
 False False False False False False False False False False  True False
  True False False False False False  True False False  True False  True
 False False  True False]
Values (no noise):  [ 1  1  1  1  1  1  1 -1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1
  1  1  1  1  1  1  1]
Logical error rate:  0.03225806451612903
```

Part 2: Implementing QEC

Obviiously in real life we do not want to directly measure the qubits and collapse them. Instead, we implemented the QEC circuit and transferred the syndromes to auxiliary logical qubit $|+\rangle_L$. By measuring the auxiliary qubit, we're able to extract the syndrome information without affecting the original quantum state $|\psi\rangle_L$. Additionally we added mechanisms to run this error correction mechanisms over multiple iterations, which allows us to implement post-selection.

We also investigated two types of noise mechanisms. The first is a simple depolarizing noise applied at the end of each iteration. The second is a simulated noise model provided by `bloqade`.

We tested our QEC circuit by measuring the $|\psi\rangle_L$ at the very end and observing a reduction in error rate by leaving out erroneous qubits.

QEC circuit with two (2) iterations and 0.1 depolarization noise:

```
q0: -SQRT_Y_DAG---@---------------@-------------DEPOLARIZE1(0.1)-----------------
                  |               |
q1: -SQRT_Y_DAG-@-|---------------@------SQRT_Y-DEPOLARIZE1(0.1)-----------------
                | |
q2: -SQRT_Y_DAG-@-|------@-SQRT_Y-@------SQRT_Y-DEPOLARIZE1(0.1)-----------------
                  |      |        |
q3: -SQRT_Y_DAG-@-@------|-SQRT_Y-@-------------DEPOLARIZE1(0.1)-----------------
                |        |
q4: -SQRT_Y_DAG-@--------|-@------SQRT_Y-@------SQRT_Y-----------DEPOLARIZE1(0.1)
                         | |             |
q5: -SQRT_Y_DAG-@--------@-|------SQRT_Y-@-----------------------DEPOLARIZE1(0.1)
                |          |
q6: ------------@-SQRT_Y---@------SQRT_Y-------------------------DEPOLARIZE1(0.1)

q7: -------------------------------------------------------------SQRT_Y_DAG------
```

```
 q8: -------------------------------------------------------------SQRT_Y_DAG------

 q9: -------------------------------------------------------------SQRT_Y_DAG------

q10: -------------------------------------------------------------SQRT_Y_DAG------

q11: -------------------------------------------------------------SQRT_Y_DAG------

q12: -------------------------------------------------------------SQRT_Y_DAG------

q13: -------------------------------------------------------------H---------------

 q0: --------------------------------------------------@---------------------------------------
                                                       |
 q1: --------------------------------------------------|----------------@----------------------
                                                       |                |
 q2: --------------------------------------------------|----------------|-@--------------------
                                                       |                | |
 q3: --------------------------------------------------|----------------|-|-@------------------
                                                       |                | | |
 q4: --------------------------------------------------|----------------|-|-|-@----------------
                                                       |                | | | |
 q5: --------------------------------------------------|----------------|-|-|-|-@--------------
                                                       |                | | | | |
 q6: --------------------------------------------------|----------------|-|-|-|-|-@------------
                                                       |                | | | | | |
 q7: ---@---------------@-------------DEPOLARIZE1(0.1)-X----------------|-|-|-|-|-|-M(0):rec[0]
        |               |                                               | | | | | |
 q8: -@-|---------------@------SQRT_Y-DEPOLARIZE1(0.1)------------------X-|-|-|-|-|-M(0):rec[1]
      | |                                                                 | | | | |
 q9: -@-|------@-SQRT_Y-@------SQRT_Y-DEPOLARIZE1(0.1)--------------------X-|-|-|-|-M(0):rec[2]
        |      |        |                                                   | | | |
q10: -@-@------|-SQRT_Y-@-------------DEPOLARIZE1(0.1)----------------------X-|-|-|-M(0):rec[3]
      |        |                                                              | | |
q11: -@--------|-@------SQRT_Y-@------SQRT_Y-----------DEPOLARIZE1(0.1)-------X-|-|-M(0):rec[4]
               | |             |                                                | |
q12: -@--------@-|------SQRT_Y-@-----------------------DEPOLARIZE1(0.1)---------X-|-M(0):rec[5]
      |          |                                                                |
q13: -@-SQRT_Y---@------SQRT_Y-------------------------DEPOLARIZE1(0.1)-----------X-M(0):rec[6]

 q0: -DEPOLARIZE1(0.1)------------------------------------------------------------

 q1: -DEPOLARIZE1(0.1)------------------------------------------------------------

 q2: -DEPOLARIZE1(0.1)------------------------------------------------------------

 q3: -DEPOLARIZE1(0.1)------------------------------------------------------------

 q4: -DEPOLARIZE1(0.1)------------------------------------------------------------

 q5: -DEPOLARIZE1(0.1)------------------------------------------------------------

 q6: -DEPOLARIZE1(0.1)------------------------------------------------------------

 q7: -R----------------SQRT_Y_DAG---@---------------@-------------DEPOLARIZE1(0.1)
                                    |               |
 q8: -R----------------SQRT_Y_DAG-@-|---------------@------SQRT_Y-DEPOLARIZE1(0.1)
                                  | |
 q9: -R----------------SQRT_Y_DAG-@-|------@-SQRT_Y-@------SQRT_Y-DEPOLARIZE1(0.1)
                                    |      |        |
q10: -R----------------SQRT_Y_DAG-@-@------|-SQRT_Y-@-------------DEPOLARIZE1(0.1)
                                  |        |
q11: -R----------------SQRT_Y_DAG-@--------|-@------SQRT_Y-@------SQRT_Y----------
                                           | |             |
q12: -R----------------SQRT_Y_DAG-@--------@-|------SQRT_Y-@----------------------
                                  |          |
q13: -R----------------H----------@-SQRT_Y---@------SQRT_Y------------------------

 q0: -@-----------------------------------------M(0):rec[14]-
      |
 q1: -|----------------@------------------------M(0):rec[15]-
      |                |
 q2: -|----------------|-@----------------------M(0):rec[16]-
      |                | |
 q3: -|----------------|-|-@--------------------M(0):rec[17]-
      |                | | |
 q4: -|----------------|-|-|-@------------------M(0):rec[18]-
      |                | | | |
 q5: -|----------------|-|-|-|-@----------------M(0):rec[19]-
      |                | | | | |
 q6: -|----------------|-|-|-|-|-@--------------M(0):rec[20]-
      |                | | | | | |
 q7: -X----------------|-|-|-|-|-|-M(0):rec[7]--R------------
                       | | | | | |
 q8: ------------------X-|-|-|-|-|-M(0):rec[8]--R------------
                         | | | | |
 q9: --------------------X-|-|-|-|-M(0):rec[9]--R------------
                           | | | |
q10: ----------------------X-|-|-|-M(0):rec[10]-R------------
                             | | |
q11: -DEPOLARIZE1(0.1)-------X-|-|-M(0):rec[11]-R------------
                               | |
q12: -DEPOLARIZE1(0.1)---------X-|-M(0):rec[12]-R------------
                                 |
q13: -DEPOLARIZE1(0.1)-----------X-M(0):rec[13]-R------------
```

In [ ]:
```python
# Part 2: QEC cycle


@squin.kernel
def msd(q, noise):
    """Parametric MSD encoding with noise"""

    squin.broadcast.sqrt_y_adj(q[:6])
    squin.broadcast.cz(IList([q[1], q[3], q[5]]), IList([q[2], q[4], q[6]]))
    squin.sqrt_y(q[6])
    squin.broadcast.cz(IList([q[0], q[2], q[4]]), IList([q[3], q[5], q[6]]))
    squin.broadcast.sqrt_y(q[2:7])
    squin.broadcast.cz(IList([q[0], q[2], q[4]]), IList([q[1], q[3], q[5]]))
    squin.broadcast.sqrt_y(IList([q[1], q[2], q[4]]))

    # noise:
    if noise > 0:
        squin.broadcast.depolarize(noise, q)


def qec_factory(*, iterations: int = 1, noise: float = 0, noise_model=None):
    """Create a QEC cycle kernel with noise"""

    @squin.kernel
    def qec():
        q = squin.qalloc(14)
        msd(q[0:7], 0)

        for _ in range(iterations):
            # apply noise to simulate decoherence over time
            if noise > 0:
                squin.broadcast.depolarize(noise, q[0:7])
            squin.h(q[13])
            msd(q[7:14], noise)
            squin.broadcast.cx(q[0:7], q[7:14])
            squin.broadcast.measure(q[7:14])

            # gotta reset the auxiliary state!
            squin.broadcast.reset(q[7:14])

        squin.broadcast.measure(q[0:7])

    if noise_model is not None:
        from bloqade.cirq_utils import load_circuit
        from bloqade.cirq_utils.emit import emit_circuit
        from bloqade.cirq_utils.noise import transform_circuit

        cirq_qec = emit_circuit(qec)
        noisy_cirq_qec = transform_circuit(cirq_qec, model=noise_model)
        return load_circuit(noisy_cirq_qec)
    return qec


circ = bloqade.stim.Circuit(qec_factory(iterations=2, noise=0.1))
circ.diagram()
```

In [ ]:
```
from bloqade.cirq_utils.noise import GeminiOneZoneNoiseModel

circ = bloqade.stim.Circuit(qec_factory(noise_model=GeminiOneZoneNoiseModel()))
circ.diagram()
```

In [6]:
```
samples = sample(circ)
values = logical_obs(samples[:, 7:14])
synds = syndromes(samples[:, 0:7])
mask = noise_free_mask(samples[:, 0:7])

print("Values: ", values)
print("Syndromes: ", synds)
print("Noise-free mask: ", mask)
print("Values (no noise): ", values[mask])
print("Logical error rate: ", logical_error(values[mask]))
```

```
Values:  [ 1  1  1  1 -1  1  1  1  1 -1  1  1  1  1  1  1  1  1  1  1  1  1  1  1
  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1
  1  1  1  1  1  1  1  1  1 -1  1  1  1  1  1  1  1  1  1  1 -1  1  1  1
  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1 -1  1  1  1
  1  1 -1  1]
Syndromes:  [array([ 1,  1,  1,  1, -1,  1,  1,  1,  1, -1,  1,  1,  1, -1,  1,  1,  1,
       -1,  1,  1,  1,  1,  1,  1,  1,  1, -1,  1,  1,  1,  1,  1, -1,  1,
        1,  1,  1,  1,  1,  1, -1,  1,  1,  1,  1,  1,  1,  1, -1,  1,  1,
        1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1, -1,  1,  1,  1,  1,  1,
       -1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,
        1,  1, -1,  1,  1,  1, -1,  1, -1,  1,  1,  1,  1, -1,  1]), array([ 1,  1,  1, -1, -1, -1, -1,  1,  1, -
1,  1,  1,  1,  1,  1,  1,  1,
       -1,  1, -1,  1,  1,  1, -1,  1,  1, -1, -1,  1, -1,  1, -1, -1,  1,
        1,  1,  1,  1,  1,  1, -1,  1,  1,  1,  1,  1,  1,  1, -1,  1,  1,
        1,  1,  1, -1,  1,  1, -1,  1,  1,  1,  1,  1, -1,  1,  1,  1,  1,
        1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1, -1,  1,  1,
        1,  1, -1,  1,  1,  1,  1,  1,  1,  1,  1,  1, -1,  1,  1]), array([ 1,  1,  1, -1, -1, -1, -1,  1,  1,
1,  1,  1,  1,  1,  1,  1, -1,
        1,  1, -1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1, -1,  1,  1,
        1, -1,  1,  1, -1,  1, -1,  1,  1,  1,  1,  1,  1,  1, -1,  1,  1,
       -1,  1,  1, -1,  1,  1, -1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,
        1,  1,  1,  1,  1,  1, -1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,
        1,  1,  1,  1,  1,  1, -1,  1,  1,  1,  1,  1, -1, -1, -1])]
Noise-free mask:  [ True  True  True False False False False  True  True False  True  True
  True False  True  True False False  True False  True  True  True False
  True  True False False  True False  True False False  True  True False
  True  True False  True False  True  True  True  True  True  True  True
 False  True  True False  True  True False  True  True False  True  True
  True  True False False  True  True  True  True False  True  True  True
  True  True False  True  True  True  True  True  True  True False  True
  True  True  True False  True  True  True False  True False  True  True
  True False False False]
Values (no noise):  [ 1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1
  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1
  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1 -1  1  1  1]
Logical error rate:  0.014925373134328358
```

In [7]:
```python
logical_error(values)
```

Out[7]:
```
0.06
```

Part 3: Post-selection

We implemented post selection by analyzing the measurement data from the QEC circuit. We calculated a binary mask that corresponds to whether the syndromes indicate the corresponding qubit is noise-free or not. We also calculated the logical error rate, physical error rate, as well as the survival rate of qubits over multiple iterations. We continue to observe a decrease in the logical error rate compared to the physical error rate. The survival rate, however, is quite a bit lower than we expected.

In [8]:
```python
# Part 3: Post-selection over multiple iterations


def post_select(*, iterations: int, shots: int = 100, verbose: bool = True, **kwargs):
    """Post-select memory qubits based on no detected errors over multiple iterations"""

    circ = bloqade.stim.Circuit(qec_factory(iterations=iterations, **kwargs))
    samples = sample(circ, shots=shots)
    values = logical_obs(samples[:, -7:])

    alive_mask = np.ones(shots, dtype=bool)
    for i in range(iterations):
        synds = syndromes(samples[:, 7 * i : 7 * (i + 1)])
        mask = noise_free_mask(samples[:, 7 * i : 7 * (i + 1)])
        alive_mask &= mask

        if verbose:
            print(f"Step {i+1}: ", "".join([str(int(s)) for s in alive_mask]))

    if verbose:
        print("Logical error: ", logical_error(values[alive_mask]))
        print("Physical error: ", logical_error(values))
        print("Survival rate: ", alive_mask.sum() / shots)

    return (
        logical_error(values[alive_mask]),
        logical_error(values),
        float(alive_mask.sum() / shots),
    )


post_select(iterations=7, noise=0.02, shots=100)
```

```
Step 1:  1111001011010110111110101011111111111001111111111111001111111111111111111111111111011111010011110111 0
Step 2:  1110001011010100111110101011111011111000010111111101001111111111111111010011111011011111000011110111 0
Step 3:  0110001010000100111110101011111001101000010110011001000111101111111111010010111010010111000011110111 0
Step 4:  0110001010000100111110101011111001001000010110011001000111101110100110100001010100101110000101101110
Step 5:  0110001010000100101110101011110001001000010110001001000011101000100110100001010100001110000101001110
Step 6:  0110001010000000101110101011110001001000010010000000100001110100010011010000101010000111000010100111 0
Step 7:  0110001010000000100010000011110001001000010000000000100001110000010011010000101010000101000010100111 0
Logical error:  0.0
Physical error:  0.17
Survival rate:  0.31
```

Out[8]: (0.0, 0.17, 0.31)

In [9]:
```python
post_select(
    iterations=5, noise_model=GeminiOneZoneNoiseModel(scaling_factor=0.1), shots=100
)
```

```
Step 1:  1111111111111101111110111011111110111111111111111111111111111110111111111111110111111111111111111011
Step 2:  1111011111111101111110111011011110110111111101111111111111111101110010111111001111111111111101001110 11
Step 3:  1111011011111101111110111011011110110101111101110111010111011011100101111110011111111111111010010101 1
Step 4:  1011011011111101110110111011011110110101110101010111010111011011100101111110011111111111111010010101 1
Step 5:  1011011011111101010110111011011110110100110001010111010111011011100101111110011111101111111010010101 1
Logical error:  0.0
Physical error:  0.04
Survival rate:  0.67
```

Out[9]: (0.0, 0.04, 0.67)

Part 4: Data Analysis

We varied parameters related to the intensity and type of noise present, as well as iterations of the QEC circuit. We simulated each combination of the parameters and collected data consisting of the error and survival rates.

The analyses of the data are provided in the presentation.

In [11]:
```python
# Part 4: Varying noise and iterations; data collection

import pandas as pd
import itertools

grid_iterations = [1, 2, 3, 5, 8, 13, 21]
grid_noise = [0.0, 0.001, 0.005, 0.01, 0.02, 0.05, 0.1, 0.2]
grid_nm_scaling = [0.01, 0.02, 0.1, 0.2, 0.5]

print("Depolarizing noise results:")
depol_results = []
for iterations, noise in itertools.product(grid_iterations, grid_noise):
    log, phys, survival = post_select(
        iterations=iterations, noise=noise, shots=1000, verbose=False
    )
    print(f"({iterations}, {noise:.3f}) => ({log:.4f}, {phys:.4f}, {survival:.4f})")
    depol_results.append([iterations, noise, log, phys, survival])

print()

print("Gemini noise model results:")
nm_results = []
```

```python
for iterations, scaling in itertools.product(grid_iterations, grid_nm_scaling):
    noise_model = GeminiOneZoneNoiseModel(scaling_factor=scaling)
    log, phys, survival = post_select(
        iterations=iterations, noise_model=noise_model, shots=1000, verbose=False
    )
    print(f"({iterations}, {scaling:.3f}) => ({log:.4f}, {phys:.4f}, {survival:.4f})")
    nm_results.append([iterations, scaling, log, phys, survival])
```

```
Depolarizing noise results:
(1, 0.000) => (0.0000, 0.0000, 1.0000)
(1, 0.001) => (0.0000, 0.0040, 0.9890)
(1, 0.005) => (0.0000, 0.0120, 0.9490)
(1, 0.010) => (0.0000, 0.0140, 0.9220)
(1, 0.020) => (0.0000, 0.0330, 0.8380)
(1, 0.050) => (0.0046, 0.0830, 0.6470)
(1, 0.100) => (0.0268, 0.1960, 0.3730)
(1, 0.200) => (0.1567, 0.3090, 0.2170)
(2, 0.000) => (0.0000, 0.0000, 1.0000)
(2, 0.001) => (0.0000, 0.0050, 0.9800)
(2, 0.005) => (0.0000, 0.0190, 0.9050)
(2, 0.010) => (0.0000, 0.0360, 0.8340)
(2, 0.020) => (0.0000, 0.0770, 0.6830)
(2, 0.050) => (0.0026, 0.1770, 0.3880)
(2, 0.100) => (0.0195, 0.2730, 0.1540)
(2, 0.200) => (0.2069, 0.3870, 0.0290)
(3, 0.000) => (0.0000, 0.0000, 1.0000)
(3, 0.001) => (0.0000, 0.0060, 0.9730)
(3, 0.005) => (0.0000, 0.0250, 0.8670)
(3, 0.010) => (0.0000, 0.0620, 0.7550)
(3, 0.020) => (0.0017, 0.1140, 0.5740)
(3, 0.050) => (0.0087, 0.2210, 0.2310)
(3, 0.100) => (0.0196, 0.3570, 0.0510)
(3, 0.200) => (0.1667, 0.4740, 0.0060)
(5, 0.000) => (0.0000, 0.0000, 1.0000)
(5, 0.001) => (0.0000, 0.0130, 0.9600)
(5, 0.005) => (0.0000, 0.0490, 0.7810)
(5, 0.010) => (0.0000, 0.0900, 0.6410)
(5, 0.020) => (0.0000, 0.1770, 0.3930)
(5, 0.050) => (0.0091, 0.3050, 0.1100)
(5, 0.100) => (0.0000, 0.4530, 0.0070)
(5, 0.200) => (1.0000, 0.4880, 0.0010)
(8, 0.000) => (0.0000, 0.0000, 1.0000)
(8, 0.001) => (0.0000, 0.0250, 0.9160)
(8, 0.005) => (0.0000, 0.0690, 0.7120)
(8, 0.010) => (0.0020, 0.1600, 0.4910)
(8, 0.020) => (0.0000, 0.2210, 0.2480)
(8, 0.050) => (0.0000, 0.4320, 0.0190)
(8, 0.100) => (nan, 0.5060, 0.0000)
(8, 0.200) => (nan, 0.4800, 0.0000)
(13, 0.000) => (0.0000, 0.0000, 1.0000)
(13, 0.001) => (0.0000, 0.0190, 0.8900)
(13, 0.005) => (0.0000, 0.1160, 0.5350)
(13, 0.010) => (0.0000, 0.1930, 0.3020)
(13, 0.020) => (0.0000, 0.3150, 0.0970)
(13, 0.050) => (0.0000, 0.4630, 0.0040)
(13, 0.100) => (nan, 0.4890, 0.0000)
(13, 0.200) => (nan, 0.5000, 0.0000)
(21, 0.000) => (0.0000, 0.0000, 1.0000)
(21, 0.001) => (0.0000, 0.0370, 0.8300)
(21, 0.005) => (0.0000, 0.1620, 0.4150)
(21, 0.010) => (0.0000, 0.2820, 0.1550)
(21, 0.020) => (0.0000, 0.4060, 0.0220)
(21, 0.050) => (nan, 0.4830, 0.0000)
(21, 0.100) => (nan, 0.5060, 0.0000)
(21, 0.200) => (nan, 0.5130, 0.0000)

Gemini noise model results:
(1, 0.010) => (0.0000, 0.0020, 0.9970)
(1, 0.020) => (0.0000, 0.0040, 0.9870)
(1, 0.100) => (0.0052, 0.0080, 0.9660)
(1, 0.200) => (0.0077, 0.0210, 0.9090)
(1, 0.500) => (0.0164, 0.0490, 0.7930)
(2, 0.010) => (0.0010, 0.0050, 0.9890)
(2, 0.020) => (0.0020, 0.0060, 0.9820)
(2, 0.100) => (0.0022, 0.0240, 0.8960)
(2, 0.200) => (0.0060, 0.0270, 0.8290)
(2, 0.500) => (0.0131, 0.0700, 0.6120)
(3, 0.010) => (0.0010, 0.0010, 0.9870)
```

```
(3, 0.020) => (0.0010, 0.0050, 0.9650)
(3, 0.100) => (0.0000, 0.0160, 0.8430)
(3, 0.200) => (0.0028, 0.0450, 0.7020)
(3, 0.500) => (0.0052, 0.1030, 0.3840)
(5, 0.010) => (0.0000, 0.0020, 0.9640)
(5, 0.020) => (0.0000, 0.0060, 0.9350)
(5, 0.100) => (0.0015, 0.0290, 0.6500)
(5, 0.200) => (0.0021, 0.0730, 0.4800)
(5, 0.500) => (0.0066, 0.1480, 0.1520)
(8, 0.010) => (0.0000, 0.0040, 0.9270)
(8, 0.020) => (0.0000, 0.0180, 0.8540)
(8, 0.100) => (0.0045, 0.0640, 0.4420)
(8, 0.200) => (0.0049, 0.1170, 0.2040)
(8, 0.500) => (0.0667, 0.2440, 0.0150)
(13, 0.010) => (0.0000, 0.0090, 0.8200)
(13, 0.020) => (0.0015, 0.0180, 0.6860)
(13, 0.100) => (0.0000, 0.0900, 0.1760)
(13, 0.200) => (0.0000, 0.1810, 0.0310)
(13, 0.500) => (nan, 0.3210, 0.0000)
(21, 0.010) => (0.0031, 0.0140, 0.6490)
(21, 0.020) => (0.0000, 0.0330, 0.3970)
(21, 0.100) => (0.0000, 0.1390, 0.0240)
(21, 0.200) => (nan, 0.2040, 0.0000)
(21, 0.500) => (nan, 0.3900, 0.0000)
```

In [12]:
```
depol_df = pd.DataFrame(
    depol_results,
    columns=[
        "iterations",
        "depol_noise",
        "logical_error",
        "physical_error",
        "survival_rate",
    ],
)
nm_df = pd.DataFrame(
    nm_results,
    columns=[
        "iterations",
        "nm_scaling",
        "logical_error",
        "physical_error",
        "survival_rate",
    ],
)

depol_df.to_csv("depol_results.csv", index=False)
nm_df.to_csv("nm_results.csv", index=False)
```