2026 QuEra Technical Challenge Presentation

QuEra
Putting Quantum to Work

Feb 1, 2026

Presented by

PhaZe Clan

Ayush B | Leo G | Krishna S | Dhruva C | Vignesh SK

Content

Today's quantum computers face issues with noise and decoherence.

We aim use the following to benchmark and improve performance of the logical qubit as a memory.

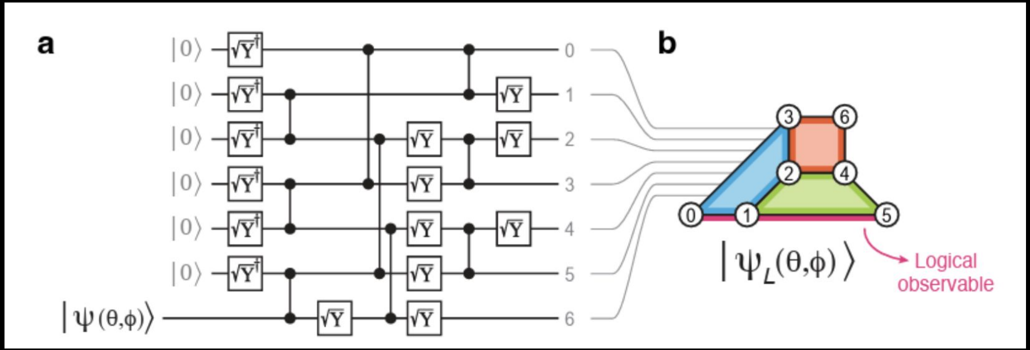
Steane QEC with [7-1-3] color code and magic state	Steane QEC with [7-1-3] color code and non-magic state	Steane QEC with [17-1-5] color code and magic state	Steane QEC with [17-1-5] color code and non-magic state

We compiled our static circuit into a Squin circuit, converted to Cirq, applied the GeminiOneZoneNoiseModel with custom noise modulation, parallelized the circuit, and re-converted to Squin for TSim

BLOQADE	Cirq	BLOQADE
Compiled circuit	Noise + Parallelize	TSim

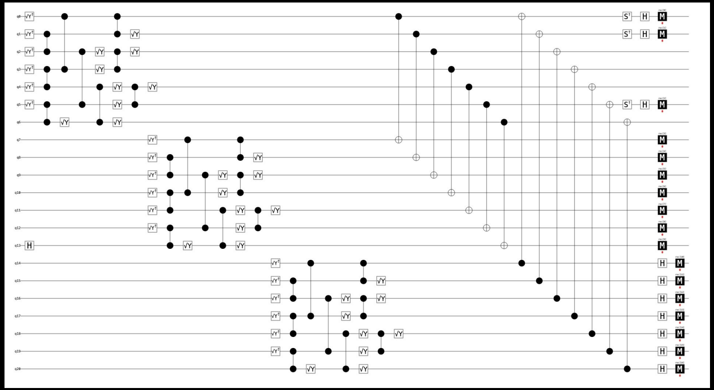
```python
def embed_mover_noise(k, noise_param, dnoise):
    """
    Apply Gemini noise to a Squin kernel.
    """
    # Convert Squin kernel to Cirq circuit
    cirq_circuit = utils.emit_circuit(k)

    # Set default noise for each parameter
    default_noises = {param: dnoise for param in noise_param}

    # Initialize Gemini noise model
    noise_model = utils.noise.GeminiOneZoneNoiseModel(**default_noises)

    # Apply noise and convert to native gates, parallelizing if possible
    noisy_cirq_circuit = utils.noise.transform_circuit(
        cirq_circuit, to_native_gateset=True, model=noise_model, parallelize_circuit=True
    )

    # Load back into Squin
    squin_circuit = utils.load_circuit(noisy_cirq_circuit, kernel_name="main_loaded")

    return squin_circuit
```

We first prepared the T state, entangled it with 6 physical qubits, and injected the resulting logical qubit into a distance-3 color code [[7-1-3]]



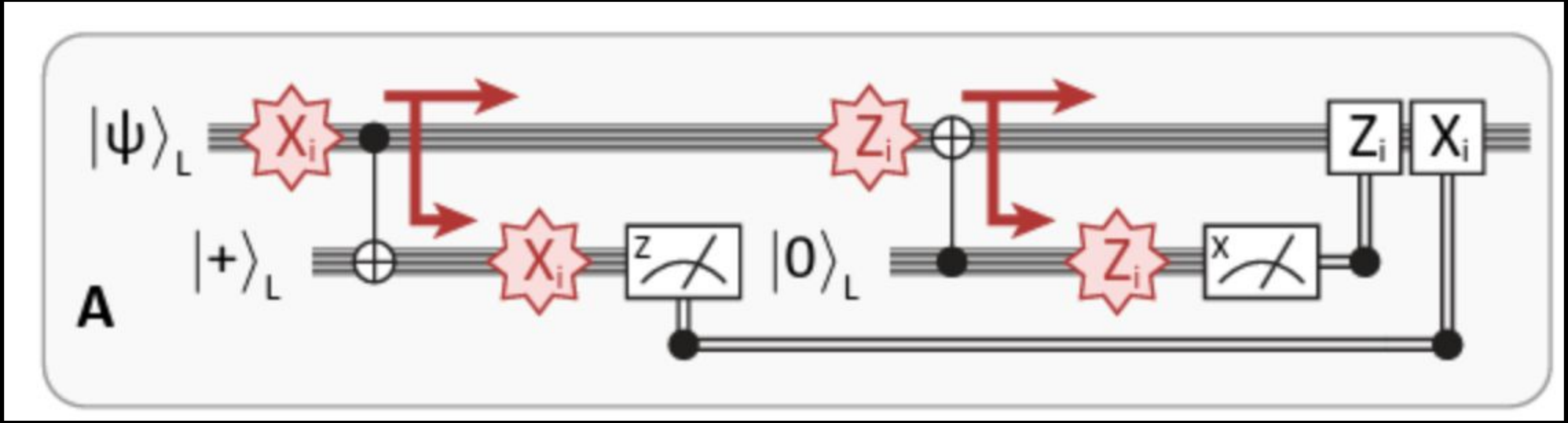
T state

Reference circuit



Code visualization of circuit

We utilized ancilla qubits to capture 2 types of unwanted noise: X (bit) flips and Z (phase) flips



Prepare ancilla qubit

Initialize to |+> and CNOT target ancilla

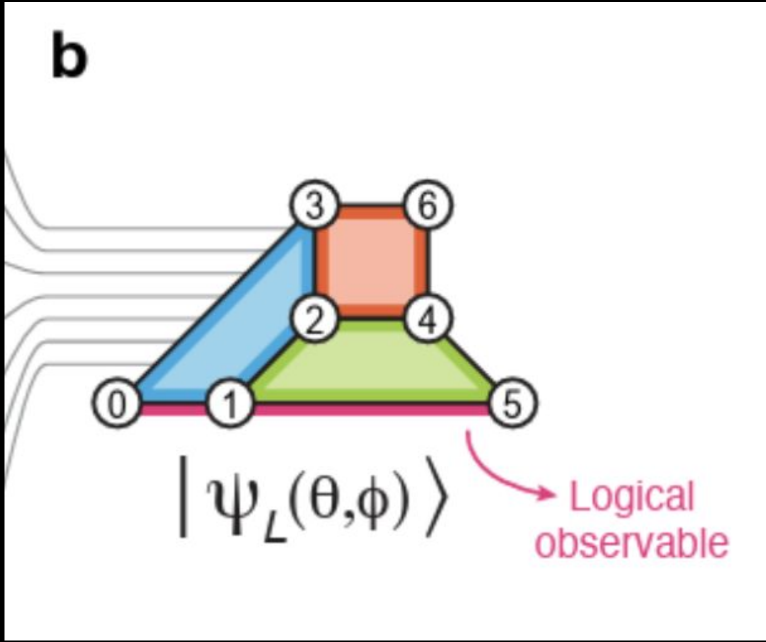
Measure in Z basis to detect X flips

Prepare ancilla qubit

Initialize to |+> and CNOT control ancilla

Measure in X basis to detect Z flips

Without collapsing the logical qubit, we obtained the stabilizers and syndromes for both ancilla to determine post-selection



Stabilizer	Covered Qubits
Stabilizer Blue	Qubits 0, 1, 2, 3
Stabilizer Red	Qubits 2, 3, 4, 6
Stabilizer Green	Qubits 1, 2, 4, 5

If the product of the eigenvalues of the covered qubits is -1, then an error has occurred in that stabilizer color zone.

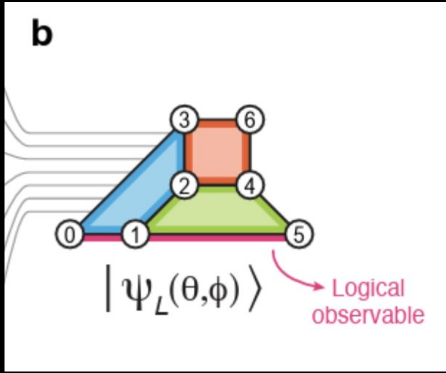
If even *one* stabilizer has eigenvalue -1 (syndrome indicates error) we throw out the entire run

We measure the shots that made it to the end using the logical observable, quantum tomography, and fidelity calculations

Inject magic state into color code → Prepare ancilla qubits to capture X, Z flips → Measure stabilizers and read syndromes → Measure shots using logical observable → Calculate fidelity (1-logical error rate)

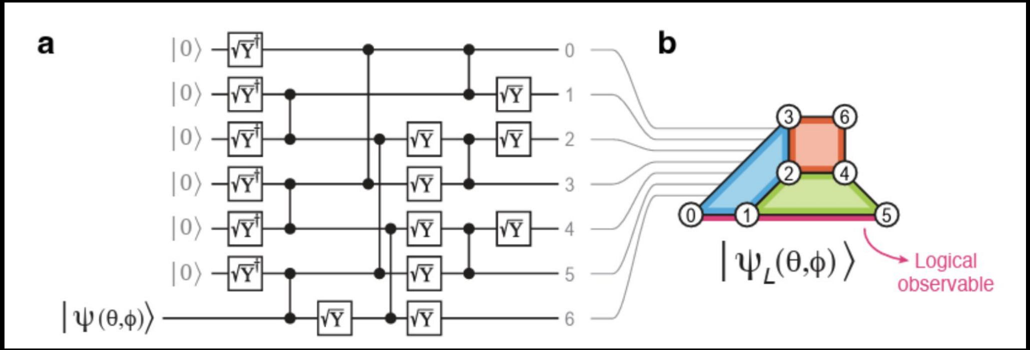
(Repeat for multiple rounds)

Throw out error-ridden shots



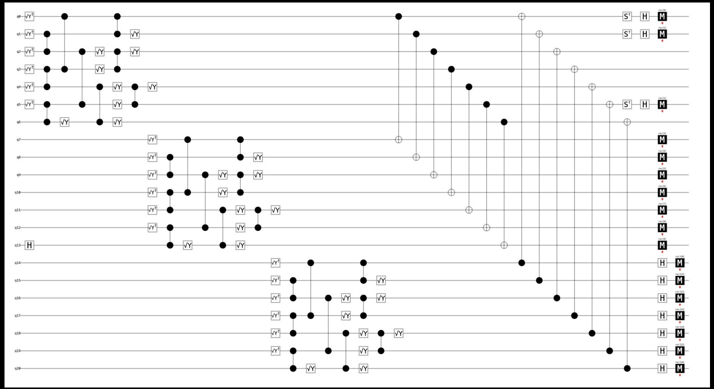
Calculating the product of the eigenvalues of the logical observable qubits gives us information to perform quantum tomography and obtain fidelity

We also injected the 0 state into a distance-3 color code to compare its logical error rates vs the previous magic T state



|0> state

Reference circuit

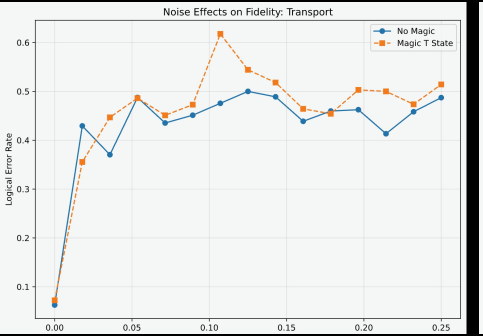


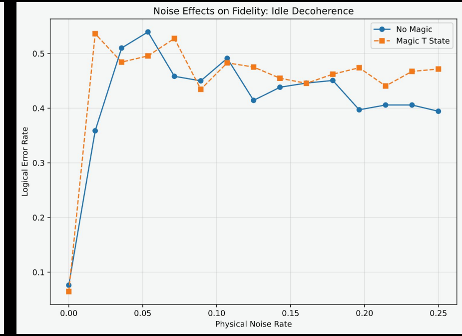
Code visualization of circuit

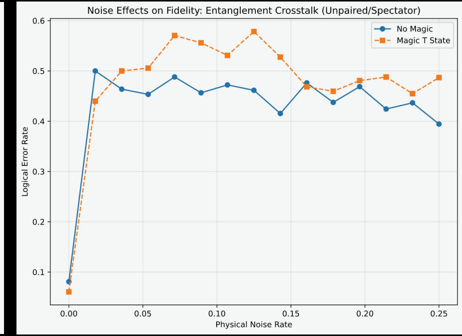
We prepared 4 relevant noise channels, grouping together related parameters presented by GeminiOneZoneNoiseModel

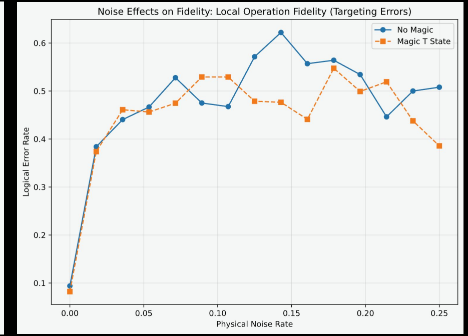
Noise Group	Significance	Parameters
Transport	Decoherence errors accumulated while atom is physically moved by optical tweezers	• mover_px • mover_py • mover_pz
Idle Decoherence	Decoherence accumulated by stationary "sitter" atoms that wait while other atoms are being moved around them	• sitter_px • sitter_py • sitter_pz
Entangled Crosstalk	Error accumulated by "spectator" atoms that sit in global laser beam without a partner during two-qubit gates	• cz_unpaired_gate_px • cz_unpaired_gate_py • cz_unpaired_gate_pz
Local Operation Fidelity	baseline error rate for single-qubit gates that require precise individual targeting (~10 times higher than global operations)	• local_px • local_py • local_pz

As expected, using our fidelity calculations from QEC, we were able to plot a roughly quadratic relationship between fidelity (1-logical error rate) as a function of physical error rate



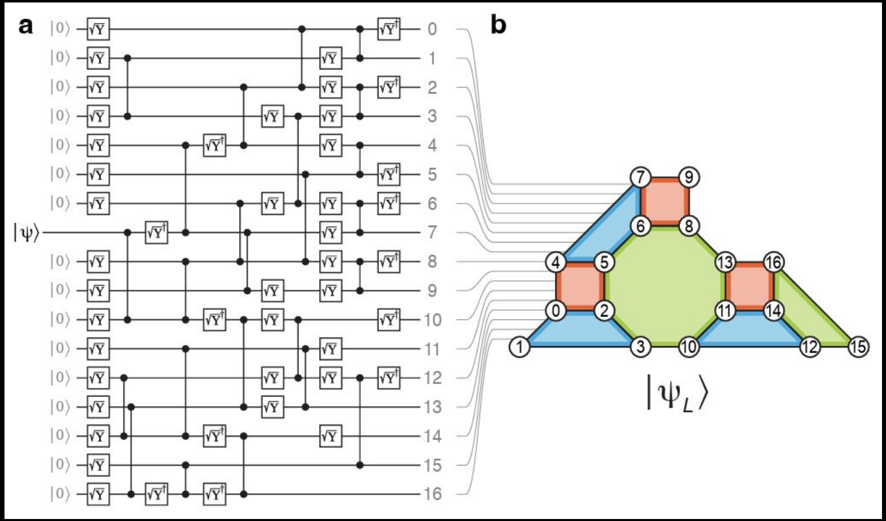






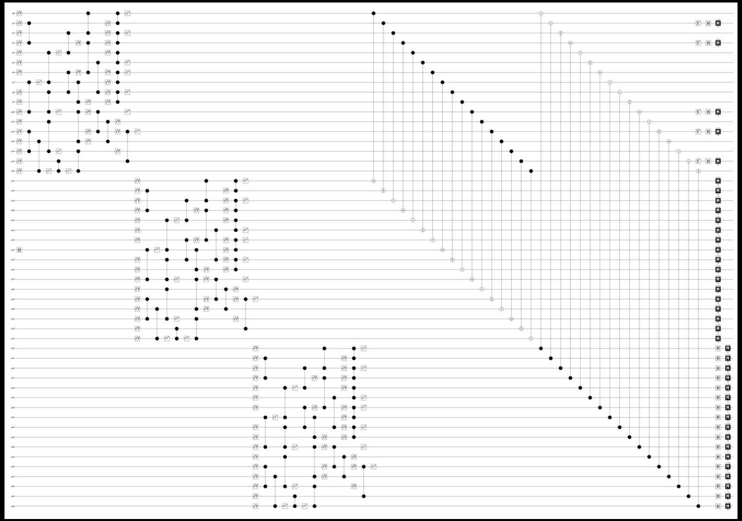
Across the 4 noise models, the non-magic state generally performed similar to the magic state (comparable logical error rate)

We injected the T state into a distance-5 color code with 16 physical qubits to observe its effect on logical error rates



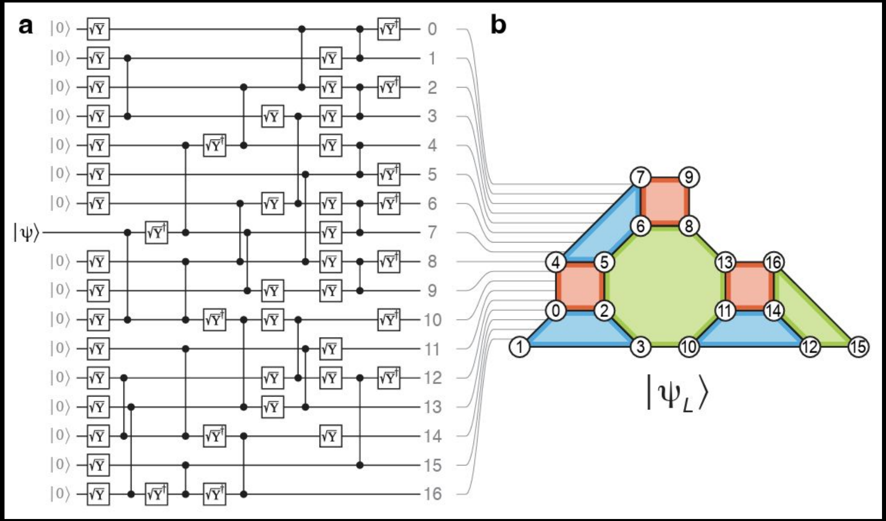
T state

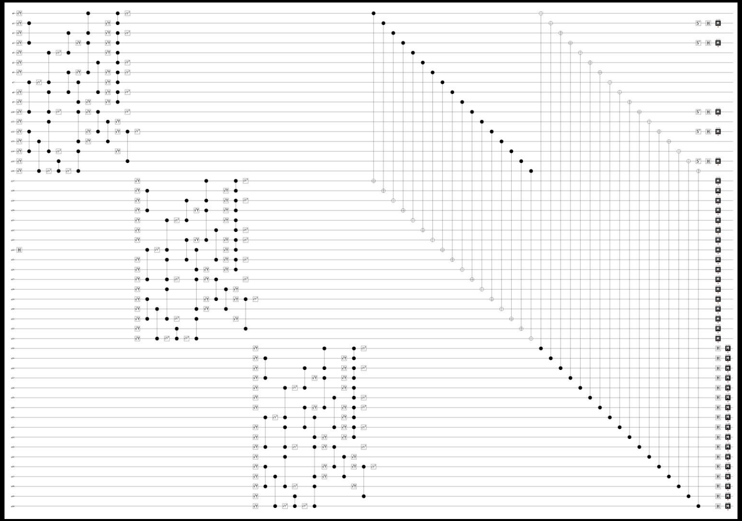
Reference circuit



Code visualization of circuit

We also injected the 0 state into a distance-5 color code to observe its effect on logical error rates



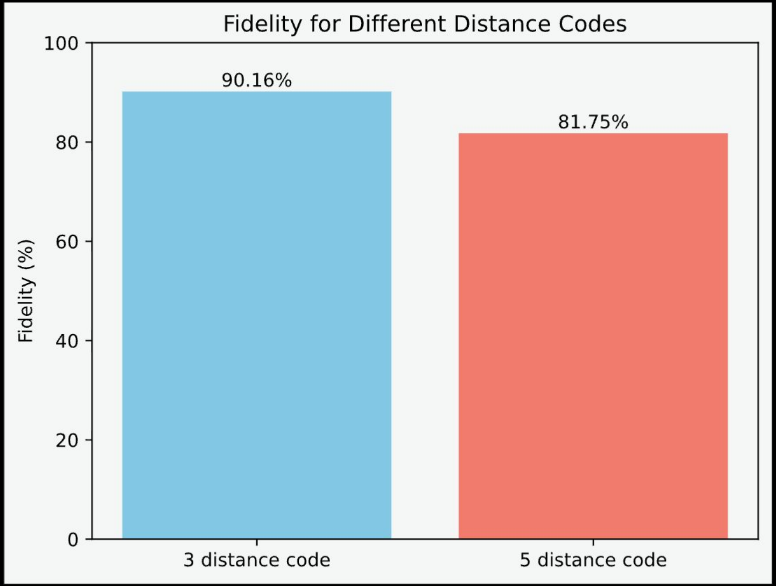


|0> state

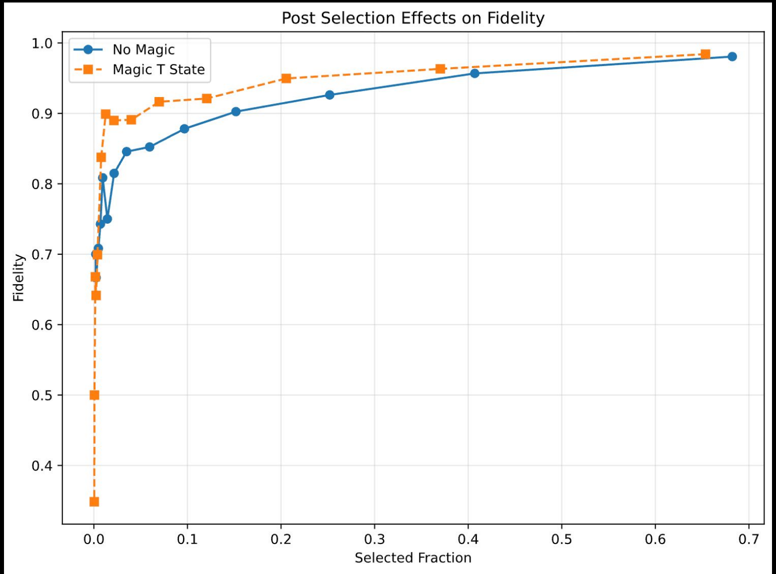
Reference circuit

Code visualization of circuit

We found an average higher fidelity for distance-3 code vs distance-5 code



Comparing our results from magic and non-magic state, we saw that higher post-selection would have the expected positive effect on fidelity for distance-5 code



While we understood the feedforward logic, we ran into a bottleneck when attempting to control quantum gates using classical bits

```python
def get_bit_and_stabilizers(kernel):
    """
    Run the kernel and extract logical measurement outcomes and stabilizers.
    """
    # Sample measurement outcomes
    samples = kernel.compile_sampler(seed=0).sample(shots=5000, batch_size=10000)

    # Split samples into logical qubits, plus block, and zero block
    logical_combination = samples[:, :3]
    plus_syndromes = samples[:, 3:10]
    zero_syndromes = samples[:, 10:17]

    # Compute logical bit values
    logical_bit = eigen_calc_prod(logical_combination)

    # Compute X-type stabilizers from + block
    plus_stabilizers = np.stack([
        eigen_calc_prod(plus_syndromes[:, [0, 1, 2, 3]]),
        eigen_calc_prod(plus_syndromes[:, [1, 2, 4, 5]]),
        eigen_calc_prod(plus_syndromes[:, [2, 4, 6, 3]])
    ], axis=1)

    # Compute Z-type stabilizers from 0 block
    zero_stabilizers = np.stack([
        eigen_calc_prod(zero_syndromes[:, [0, 1, 2, 3]]),
        eigen_calc_prod(zero_syndromes[:, [1, 2, 4, 5]]),
        eigen_calc_prod(zero_syndromes[:, [2, 4, 6, 3]])
    ], axis=1)

    # Combine X and Z stabilizers
    stabilizers = np.concatenate([plus_stabilizers, zero_stabilizers], axis=1)

    return logical_bit, stabilizers
```

Syndrome Extraction

```python
    @kernel
    def circuit():
        # Allocate qubits for logical, plus, and zero blocks
        q_logical = squin.qalloc(7)
        q_plus = squin.qalloc(7)
        q_zero = squin.qalloc(7)

        # Inject magic state if requested
        if inject_magic:
            squin.broadcast.h(q_logical[-1])
            squin.broadcast.t_adj(q_logical[-1])

        # Prepare auxiliary + state
        for _ in range(n_iterations):
            squin.broadcast.h(q_plus[-1])
            encode_block(q_logical)
            encode_block(q_plus)
            encode_block(q_zero)

            # Entangle logical, plus, and zero blocks
            squin.broadcast.cx(q_logical, q_plus)
            squin.broadcast.cx(q_zero, q_logical)

        # Measure blocks
        measure_all(q_plus)
        squin.broadcast.h(q_zero)
        measure_all(q_zero)

        # Classical logic involving if statements
        # <===== Here ======>

        measure_subset(q_logical)

    return circuit
```

Quantum Circuit Classical Logic Integration