AEGIS Project

Stage-N: Hybrid Quantum-Classical Token Execution Layer

Formal Specification for RiftLang Protocol Stack

Version 1.0 - Technical Specification Document

OBINexus Computing Division

`Toolchain: riftlang.exe` → `.so.a` → `rift.exe` → `gosilang`

July 2, 2025

Contents

1 Executive Summary

This document formalizes Stage-N of the RiftLang Protocol Stack, establishing the critical interface between quantum probabilistic computation and classical deterministic execution. Stage-N enables seamless transitions between quantum superposition states and classical computational models while maintaining strict AEGIS governance compliance throughout the quantum-classical boundary.

The specification defines standardized patterns for stages 0 through N+1 (currently implemented through Stage-7), providing a unified framework for quantum token management, collapse operations, and memory-governed parsing within the RIFT domain-specific language ecosystem.

2 Stage Evolution Framework

Stage Progression Model

- **Stage-0**: Token initialization and classical baseline
- **Stage-1**: Quantum extension introduction
- **Stage-2**: Entanglement protocol establishment
- **Stage-3**: Collapse operator implementation
- **Stage-4**: Memory governance integration
- **Stage-5**: Parser unification
- **Stage-6**: AEGIS phase alignment
- **Stage-7**: Full quantum-classical bridge deployment
- **Stage-N**: Dynamic stage instantiation
- **Stage-N+1**: Future extensibility framework

3 Core Architecture and Purpose

3.1 Fundamental Design Principles

Stage-N serves as the quantum-classical computation bridge within the RIFT DSL execution pipeline. The architecture enables:

1. Quantum state preservation during computation
2. Deterministic resolution when measurement occurs
3. Governance enforcement at state transitions
4. Memory-bounded execution with Planck-scale constraints

3.2 Integration Points

```
@stage_interface[N] {
    input_stages: [N-1, quantum_init]
    output_stages: [N+1, classical_exec]
    governance_hooks: AEGIS_Phase_I_III
    memory_model: quantum_foam_temporal
    compliance_level: STRICT
}
```

Listing 1: Stage-N Integration Architecture

4 Quantum Token Specification

4.1 Core Quantum Token Definition

The fundamental quantum token represents a qubit in superposition state with complex amplitude coefficients.

```
@token[quantum] qbit superposition(α, β) → QINT
Where:
    α in C : Complex amplitude for |0⟩ state
    β in C : Complex amplitude for |1⟩ state
    Constraint: |α|² + |β|² = 1 (normalization)
```

Listing 2: Quantum Token Base Definition

4.2 Extended Quantum Token Attributes

```
@token_extension[quantum] {
    # Dirac notation representation
    bra_ket_notation: |ψ⟩ = α |0⟩ + β |1⟩

    # Normalization enforcement with tolerance
    amplitude_norm: enforce(|α|² + |β|² = 1.0 +/- ϵ)
    Where: ϵ = 10^-15 (machine ϵ)

    # Decoherence threshold at Planck scale
    decohere_threshold: τ_planck = 5.39 x 10^-44 seconds

    # Entanglement tracking
    entanglement_flag: bool
    entanglement_partners: QINT[] (max_size = 6)

    # Phase coherence bounds
    phase_coherence: ϕ in [0, 2*π]
    phase_drift_rate: d_ϕ/dt ≤ π/τ_coherence
}
```

Listing 3: Quantum Token Extensions

4.3 Quantum Token Memory Alignment

```
@memory_align[quantum] {
    alignment: 8-qubit boundary
    span_type: distributed_quantum_foam
    coherence_window: planck_time
    isolation_level: phase_locked

    layout: {
         |q0⟩ |q1⟩ |q2⟩ |q3⟩ |q4⟩ |q5⟩ |q6⟩ |q7⟩
        [amplitude_real][amplitude_imag]
        [phase][entangle_mask][coherence]
    }
}
```

Listing 4: Memory Alignment Specification

5 Classical Resolution Operator

5.1 Collapse Operator Definition

The collapse operator manages the quantum-to-classical transition under governance constraints.

```
@operator collapse {
    input: QINT                    # Quantum integer in
       superposition
    condition: coherence ≥ PLANCK_THRESHOLD
    output: INT                    # Classical deterministic
       integer
    audit: quantum_event_log       # Governance audit trail

    properties: {
        irreversible: true
        measurement_basis: computational
        entropy_increase: Δ_S > 0
    }
}
```

Listing 5: Collapse Operator Specification

5.2 Piecewise Collapse Logic

```
PROCEDURE quantum_classical_collapse(q: QINT, E: energy, m:
   mass):
    LET c² = speed_of_light²
    LET E_planck = ℏ * ω

    IF E² ≤ m * c²:
        # Low energy - classical behavior dominates
```

```
        state := CLASSICAL_DETERMINISTIC
        RETURN cast_to_int(q, method="measurement")

    ELIF E > E_critical AND q.coherence ≥ τ_planck:
        # High energy with coherence - forced collapse
        TRIGGER collapse_event {
            log: quantum_event_log
            timestamp: current_planck_time
            method: "forced_decoherence"
        }
        state := CLASSICAL_COLLAPSED
        RETURN probabilistic_cast(q)

    ELSE:
        # Maintain quantum superposition
        state := QUANTUM_SUPERPOSITION
        EVOLVE q WITH hamiltonian(H)
        RETURN q  # Preserve quantum state

    END IF
END PROCEDURE
```

Listing 6: Collapse Decision Tree

5.3 State Transition Matrix

```
@state_transition_matrix {
    QUANTUM → CLASSICAL: {
        trigger: measurement OR decoherence
        probability: | ⟨ψ| φ>|²
        governance: collapse_contract
        audit_level: MANDATORY
    }

    CLASSICAL → QUANTUM: {
        trigger: superposition_gate
        condition: coherence_budget > threshold
        governance: quantum_init_contract
        audit_level: STRICT
    }

    QUANTUM → QUANTUM: {
        trigger: unitary_evolution
        operator: U = exp(-iHt/ℏ)
        governance: evolution_contract
        audit_level: PERIODIC
    }
}
```

Listing 7: Quantum-Classical State Transitions

6 Memory-Governed Quantum Parser

6.1 Hybrid Token Grammar

The parser must handle both quantum and classical tokens with appropriate type safety.

```
@parser[hybrid_quantum_classical] {
    token_types: {
        QINT    : quantum_integer[superposition]
        INT     : classical_integer[deterministic]
        FLOAT   : classical_float[ieee754]
        BRA     :  ⟨ψ|   quantum_state
        KET     :  |ψ⟩   quantum_state
        QFLOAT  : quantum_float[superposition]
    }

    # Regex pattern with quantum extensions
    parse_rules: R"/([QC])(INT|FLOAT|STATE)/gmi[tb]"
    Where:
        g: global matching
        m: multiline quantum states
        i: case-insensitive operators
        t: top-down classical resolution
        b: bottom-up quantum composition
}
```

Listing 8: Token Type Definitions

6.2 Temporal Memory State Management

```
@memory_state::quantum_foam {
    lifetime: planck_time = 5.39 x  10^-44  seconds
    scope: local_superposition

    allocation: {
        classical_mode: align(4096_bits)
        quantum_mode: align(8_qubits)
        hybrid_mode: interleaved_coherent
    }

    persistence: {
        coherent_duration: τ_coherence
        decoherence_rate: Γ = 1/T₂
        error_threshold: 10^-9
    }

    governance: {
        max_entanglement_depth: 6
        bell_state_limit: 4_pairs
        gc_policy: phase_aware_collection
    }
```

```
}
```

Listing 9: Quantum Memory Management

6.3 Parser State Machine

```
AUTOMATON quantum_parser {
    states: {S_INIT, S_QUANTUM, S_CLASSICAL, S_COLLAPSE,
       S_MEASURE}

    transitions: {
        S_INIT → S_QUANTUM:
            condition: detect_superposition_token()
            action: init_quantum_context()

        S_QUANTUM → S_COLLAPSE:
            condition: coherence < PLANCK_THRESHOLD
            action: prepare_collapse()

        S_COLLAPSE → S_CLASSICAL:
            condition: collapse_complete()
            action: emit_classical_token()

        S_QUANTUM → S_MEASURE:
            condition: measurement_operator()
            action: von_neumann_projection()

        S_MEASURE → S_CLASSICAL:
            condition: measurement_complete()
            action: emit_measured_value()
    }

    error_states: {
        E_COHERENCE_LOST: recovery = forced_collapse
        E_ENTANGLE_VIOLATION: recovery = isolate_subsystem
        E_MEMORY_OVERFLOW: recovery = quantum_gc
    }
}
```

Listing 10: Quantum Parser Automaton

7 Governance Constraint Declarations

7.1 Core Governance Rules

```
@gov_rule::collapse_contract {
    requires: {
        coherence ≥ planck_threshold
        entanglement_depth ≤ max_allowed
```

```
            quantum_budget > operation_cost
            audit_trail.enabled = true
        }

        prohibits: {
            superposition_state > max_density_matrix_size
            concurrent_measurements > 1
            phase_drift > π/4
            untracked_entanglement = true
        }

        audit: {
            log_destination: quantum_event_log
            retention_period: 7_stages
            cryptogra϶c_seal: SHA3-256
            immutability: blockchain_anchored
        }
    }
```

Listing 11: Collapse Contract Governance

7.2 Resource Management Constraints

```
@gov_rule::quantum_resource_management {
    allocation_policy: {
        max_qubits_per_token: 16
        max_entangled_pairs: 8
        decoherence_budget: 1000_planck_times
        memory_quota: 1MB_quantum_foam
    }

    cleanup_policy: {
        auto_collapse_timeout: 100_planck_times
        garbage_collection: phase_aware
        memory_reclaim: immediate
        entanglement_pruning: depth_first
    }

    cost_model: {
        superposition_cost: 0.1_per_qubit_per_cycle
        entanglement_cost: 0.3_per_pair
        measurement_cost: 0.2_per_operation
        coherence_maintenance: 0.05_per_planck_time
    }
}
```

Listing 12: Quantum Resource Governance

7.3 Security and Validation Rules

```
@gov_rule::quantum_security {
    validation: {
        state_vector_normalization: continuous
        no_cloning_enforcement: strict
        basis_state_verification: periodic(10_cycles)
        bell_inequality_check: on_entanglement
    }

    access_control: {
        quantum_state_read: privileged_only
        collapse_trigger: authorized_operators
        entanglement_create: rate_limited(10/sec)
        phase_manipulation: governance_approved
    }

    integrity: {
        checksum_algorithm: quantum_hash_SHA3Q
        tamper_detection: bell_inequality_test
        audit_trail: immutable_quantum_ledger
        replay_protection: nonce_per_operation
    }
}
```

Listing 13: Quantum Security Governance

8 Integration with AEGIS Phase Architecture

8.1 Phase I - Matrix Parity Integration

```
INTEGRATION matrix_parity_bridge {
    quantum_to_fft: {
        INPUT: QINT[superposition]
        PROCESS:
            1. Extract amplitude vectors (α, β)
            2. Map to FFT basis: F( |ψ⟩ ) = Sum(α_k*e^(2*π*ijk/
               N))
            3. Apply parity constraints from Phase I
            4. Verify matrix eigenvalue stability
        OUTPUT: FFT_MATRIX[classical]
    }

    governance: {
        parity_check: R"/[01]{8}/g"
        matrix_alignment: 8x8_quantum_block
        eigenvalue_threshold: |λ| < 1.0
    }
}
```

Listing 14: Matrix Parity Bridge

8.2 Phase II - Token Stream Management

```
INTEGRATION token_stream_quantum {
    stream_mode: {
        classical: sequential_ordered
        quantum: parallel_superposed
        hybrid: context_switched
    }

    synchronization: {
        barrier: quantum_measurement_point
        ordering: causal_cone_preservation
        latency: ≤ coherence_window
    }

    buffering: {
        quantum_buffer: circular_phase_locked
        classical_buffer: FIFO_deterministic
        transition_buffer: copy_on_collapse
    }
}
```

Listing 15: Token Stream Integration

8.3 Phase III - Planck Verification

```
INTEGRATION planck_verification {
    collapse_window: {
        detection: coherence < PLANCK_THRESHOLD
        action: enforce(collapse_contract)
        verification: cryptograφc_proof
        timing: exact_planck_time
    }

    quantum_classical_boundary: {
        transition_log: {
            timestamp: planck_time_resolution
            state_before:  |ψ⟩
            state_after: classical_value
            entropy_change: Δ_S
            information_preserved: I = -Sum(p log p)
        }
    }

    entanglement_boundary: {
        max_distance: 6_hops
        isolation_enforcement: bell_state_collapse
        audit_requirement: full_trace
        correlation_preservation: EPR_compliant
    }
}
```

Listing 16: Planck Scale Verification

9 Runtime Execution Model

9.1 Dual-Mode Execution Pipeline

```
PIPELINE rift_stage_n_execution {
    MODE classical {
        stages: tokenize → parse → validate → execute
        memory: sequential_4096_aligned
        concurrency: mutex_protected
        error_handling: exception_based
    }

    MODE quantum {
        stages: superpose → entangle → evolve → measure
        memory: distributed_8qubit_foam
        concurrency: phase_locked_parallel
        error_handling: decoherence_recovery
    }

    MODE hybrid {
        stages: detect_context → switch_mode → process →
           reconcile
        memory: adaptive_alignment
        concurrency: quantum_classical_barrier
        error_handling: graceful_degradation
    }

    performance: {
        classical_throughput: 10^6 ops/sec
        quantum_coherence: 1000 x planck_time
        transition_overhead: < 1us
    }
}
```

Listing 17: Stage-N Execution Pipeline

9.2 Context Switching Protocol

```
PROTOCOL context_switch {
    classical_to_quantum: {
        save_classical_state()
        init_quantum_registers()
        prepare_superposition()
        verify_coherence()
        enable_quantum_operations()
```

```
    }

    quantum_to_classical: {
        prepare_measurement()
        collapse_superposition()
        extract_classical_value()
        cleanup_quantum_resources()
        restore_classical_context()
    }

    switch_overhead: {
        time_cost: O(n_qubits)
        memory_cost: O(2^n_qubits)
        governance_cost: O(audit_depth)
    }
}
```

Listing 18: Quantum-Classical Context Switch

10 Example Usage Patterns

10.1 Basic Quantum Token Operations

```
# Initialize quantum token in superposition
@quantum
let q_value: QINT = superposition(0.707, 0.707)  # |+> state

# Entangle two quantum tokens
@quantum
let q_pair: (QINT, QINT) = entangle(q_value, q_other)

# Conditional collapse based on coherence
@hybrid
if coherence(q_value) < PLANCK_THRESHOLD {
    let classical_result: INT = collapse(q_value)
    process_classical(classical_result)
} else {
    evolve_quantum(q_value, hamiltonian)
}

# Measurement with basis selection
@quantum
let measured: INT = measure(q_value, basis="X")
```

Listing 19: Quantum Token Usage Examples

10.2 Hybrid Computation Pattern

```
@hybrid_algorithm grover_search {
```

```
    # Classical preprocessing
    let dataset: INT[] = load_classical_data()
    let target: INT = define_search_target()

    # Quantum acceleration phase
    @quantum {
        let q_register: QINT[] = init_superposition(size=log2(dataset.length))

        repeat sqrt(dataset.length) times {
            apply_oracle(q_register, target)
            apply_diffusion(q_register)

            if coherence_degraded(q_register) {
                refresh_quantum_state(q_register)
            }
        }

        let result_index: INT = measure_all(q_register)
    }

    # Classical verification
    @classical {
        verify_result(dataset[result_index], target)
        return dataset[result_index]
    }
}
```

Listing 20: Hybrid Quantum-Classical Algorithm

11 Formal Verification Properties

11.1 Safety Properties

```
PROPERTY quantum_safety {
    # G1: Normalization is always maintained
    [] (forall q: QINT . |amplitude(q)|^2 = 1.0 +/- ϵ)

    # G2: No-cloning theorem is preserved
    [] (not exists operation . clone( |ψ⟩ ) → |ψ⟩  |ψ⟩ )

    # G3: Causality is respected
    [] (forall measurement . timestamp(cause) < timestamp(effect))

    # G4: Measurement irreversibility
    [] (collapsed(q) → not quantum(q))
}
```

Listing 21: Quantum Safety Invariants

11.2 Liveness Properties

```
PROPERTY quantum_liveness {
    # L1: Every quantum state eventually decoheres
    <> (forall q: QINT . coherence(q) < PLANCK_THRESHOLD)

    # L2: Measurements eventually complete
    (measure_initiated(q) → <> measure_complete(q))

    # L3: Resources are eventually reclaimed
    []<> (allocated_qubits = deallocated_qubits)

    # L4: Entanglements eventually resolve
    <> (forall entangled_pair . separated OR measured)
}
```

Listing 22: Quantum Liveness Guarantees

12 Performance Specifications

Performance Requirements

- **Quantum Coherence Time**: $\geq 1000\tau_{planck}$
- **State Preparation**: $<$ 10 ns per qubit
- **Measurement Latency**: $<$ 100 ns
- **Context Switch Overhead**: $< 1\mu$s
- **Memory Efficiency**: $O(n)$ for n qubits
- **Entanglement Depth**: Maximum 6 levels
- **Error Rate**: $< 10^{-9}$ per operation

13 Firmware Integration Guidelines

13.1 Git-RAF Integration Points

```
@firmware_integration {
    git_raf_hooks: {
        pre_commit: validate_quantum_governance()
        post_merge: verify_collapse_consistency()
        pre_push: audit_quantum_operations()
    }

    versioning: {
        quantum_state_snapshot: on_commit
        collapse_history: immutable_log
```

```
            entanglement_graph: version_tracked
        }

        deployment: {
            stage: [0..N+1]
            firmware_target: quantum_coprocessor
            validation_level: AEGIS_COMPLIANT
        }
}
```

Listing 23: Git-RAF Firmware Hooks

14 Conclusion and Future Extensions

Stage-N of the RiftLang Protocol Stack provides a robust foundation for hybrid quantum-classical computation within the AEGIS governance framework. The specification enables:

- Seamless quantum-classical transitions
- Governance-compliant state management
- Planck-scale temporal constraints
- Integration with existing AEGIS phases
- Extensibility for future quantum algorithms

Future stages (N+1 and beyond) will extend this framework to support:

- Distributed quantum computation
- Fault-tolerant quantum error correction
- Advanced entanglement protocols
- Quantum machine learning integration

A Glossary of Terms

QINT Quantum Integer - A quantum register holding superposition states

Planck Time $\tau_{planck} = 5.39 \times 10^{-44}$ seconds

Coherence Quantum state phase relationship preservation

Collapse Quantum to classical state transition

AEGIS Automated Enterprise Governance Intelligence System

B References

1. AEGIS Project Technical Specification v2.0
2. RiftLang Compiler Documentation
3. Quantum Computing Governance Framework
4. Git-RAF Firmware Integration Manual
5. OBINexus Toolchain Architecture Guide