Fusion Control Software Suite

Functional Specification Document

Version 1.0

Recognition Science Research
engineering@recognitionscience.org

January 18, 2026

Abstract

This document specifies the functional requirements for a suite of software modules designed for inertial confinement fusion (ICF) reactor control and optimization. The suite comprises four core components: (1) **Reaction Network Optimizer** for computing optimal fusion fuel chains, (2) **φ-Scheduler Engine** for generating interference-minimized pulse timings, (3) **Symmetry Verifier** for certifying control adjustment safety, and (4) **Jitter Simulator** for quantifying hardware tolerance budgets. All modules are backed by formally verified mathematical foundations in Lean 4, enabling unprecedented confidence in safety-critical fusion applications.

Contents

1 Introduction

1.1 Purpose

This Functional Specification Document (FSD) defines the requirements, interfaces, and behaviors of the Fusion Control Software Suite. The document serves as:

- A contract between specification and implementation
- A basis for verification and validation testing
- A reference for regulatory approval submissions
- Documentation for operators and maintenance personnel

1.2 Scope

The software suite covers:

1. Pre-shot fuel optimization (Reaction Network Optimizer)
2. Pulse timing generation (φ-Scheduler Engine)
3. Real-time control verification (Symmetry Verifier)
4. Hardware specification analysis (Jitter Simulator)

1.3 Definitions and Acronyms

Term	Definition
ICF	Inertial Confinement Fusion
φ	Golden Ratio $= \frac{1+\sqrt{5}}{2} \approx 1.618$
$S(Z, N)$	Stability Distance for nucleus (Z, N)
$J(r)$	Symmetry Ledger cost functional
Magic Number	Element of $\{2, 8, 20, 28, 50, 82, 126\}$
Doubly-Magic	Nucleus with both Z and N magic

1.4 References

1. `IndisputableMonolith/Fusion/ReactionNetwork.lean` — Formal proofs
2. `IndisputableMonolith/Fusion/InterferenceBound.lean` — φ-scheduling proofs
3. `IndisputableMonolith/Fusion/LocalDescent.lean` — Symmetry verifier proofs
4. `IndisputableMonolith/Fusion/JitterRobustness.lean` — Jitter analysis proofs

2 System Overview

2.1 Architecture

The Fusion Control Software Suite consists of four independent modules that can be used standalone or integrated:

Module	Phase	Frequency	Criticality
Reaction Network Optimizer	Pre-shot	Daily/Weekly	Design
φ-Scheduler Engine	Pre-shot	Per-shot	Operational
Symmetry Verifier	Real-time	Per-adjustment	Safety-Critical
Jitter Simulator	Design	As-needed	Design

2.2 Data Flow

```
[Fuel Inventory] --> [Reaction Network Optimizer] --> [Optimal Fuel Mix]
                                                            |
                                                            v
[Timing Parameters] --> [Phi-Scheduler Engine] --> [Pulse Schedule]
                                                            |
                                                            v
[Sensor Data] --> [Symmetry Verifier] --> [Certified Adjustment]
                                                            |
                                                            v
[Hardware Specs] --> [Jitter Simulator] --> [Tolerance Budget]
```

2.3 External Interfaces

2.3.1 Hardware Interfaces

- Laser timing system (digital trigger outputs)
- Beam diagnostic sensors (intensity measurements)
- Control actuators (power adjustments)

2.3.2 Software Interfaces

- Facility control system (EPICS or equivalent)
- Nuclear data libraries (ENDF/B format)
- Logging and audit systems

3 Module 1: Reaction Network Optimizer

3.1 Purpose

Compute optimal fusion fuel combinations by analyzing the weighted reaction network graph, minimizing Stability Distance to maximize energy yield.

3.2 Functional Requirements

Requirement 1 (RNO-001: Graph Construction)**.** *The system shall construct a fusion reaction network graph from:*

- *Input: Set of available nuclear species (fuel inventory)*
- *Output: Weighted directed graph* $G = (V, E, w)$

where vertices are nuclear configurations and edges are fusion reactions.

Requirement 2 (RNO-002: Stability Distance Computation)**.** *For each nuclear configuration (Z, N), the system shall compute:*

$$S(Z, N) = d(Z) + d(N) \tag{1}$$

where $d(x) = \min_{m \in \mathcal{M}} |x - m|$ and $\mathcal{M} = \{2, 8, 20, 28, 50, 82, 126\}$.

Requirement 3 (RNO-003: Edge Weight Assignment)**.** *For each fusion reaction edge $e : (Z_1, N_1) + (Z_2, N_2) \to (Z_3, N_3)$:*

$$w(e) = S(Z_1, N_1) + S(Z_2, N_2) - S(Z_3, N_3) \tag{2}$$

Requirement 4 (RNO-004: Path Optimization)**.** *The system shall find the path from initial fuel to doubly-magic products that maximizes cumulative edge weight.*

Requirement 5 (RNO-005: Magic-Favorable Filter)**.** *The system shall optionally filter to show only Magic-Favorable reactions ($w > 0$).*

3.3 Inputs

Parameter	Type	Units	Description
`fuel_inventory`	List[(Z, N, count)]	—	Available nuclei
`max_A`	Integer	amu	Maximum mass number to consider
`target_products`	List[(Z, N)]	—	Desired end products
`filter_favorable`	Boolean	—	Show only $w > 0$ edges

3.4 Outputs

Output	Type	Description
`optimal_path`	List[Edge]	Sequence of reactions
`total_weight`	Float	Cumulative stability improvement
`fuel_recommendation`	Dict	Suggested fuel mixture
`doubly_magic_products`	List[(Z, N)]	Predicted end states

3.5 Algorithms

3.5.1 Graph Construction

```
structure Node where
  Z : Nat   -- protons
  N : Nat   -- neutrons
  deriving DecidableEq

structure Edge where
  source1 : Node
  source2 : Node
  target  : Node
  h_Z : target.Z = source1.Z + source2.Z
  h_N : target.N = source1.N + source2.N
```

3.5.2 Path Search

Dijkstra's algorithm with negated weights for maximum-weight path:

1. Initialize priority queue with starting nodes
2. For each node, explore outgoing edges
3. Update path weights and predecessors
4. Terminate when doubly-magic sink is reached

3.6 Performance Requirements

Metric	Requirement
Graph construction time	< 1 second for $A \leq 250$
Path optimization time	< 5 seconds
Memory usage	< 1 GB

3.7 Verification

The module shall pass the following test cases:

1. **He-4 terminus:** pp-chain terminates at $(2, 2)$
2. **O-16 capture:** α-capture reaches $(8, 8)$
3. **Ca-40 ladder:** α-ladder reaches $(20, 20)$

4 Module 2: φ-Scheduler Engine

4.1 Purpose

Generate pulse timing sequences using Golden Ratio intervals to minimize interference and maximize jitter robustness.

4.2 Functional Requirements

Requirement 6 (PSE-001: φ-Sequence Generation)**.** *The system shall generate pulse times according to:*

$$t_k = \tau_0 \cdot \varphi^{k-1}, \quad k = 1, \ldots, n \tag{3}$$

where τ_0 is the base timing and $\varphi = \frac{1+\sqrt{5}}{2}$.

Requirement 7 (PSE-002: Interference Ratio Bound)**.** *The generated sequence shall satisfy:*

$$R = \frac{I_{total}}{I_{self}} < \rho \tag{4}$$

for user-specified threshold ρ.

Requirement 8 (PSE-003: Duration Constraint)**.** *The system shall support specifying:*

- *Total sequence duration T*

- *Number of pulses n*

and compute τ_0 accordingly.

Requirement 9 (PSE-004: Hardware Format Export)**.** *The system shall export timing sequences in formats compatible with:*

- *TTL trigger generators*
- *Arbitrary waveform generators*
- *FPGA timing systems*

Requirement 10 (PSE-005: Visualization)**.** *The system shall provide graphical display of:*

- *Pulse timing diagram*
- *Interference overlap visualization*
- *Comparison to equal-spaced baseline*

4.3 Inputs

Parameter	Type	Units	Description
`n_pulses`	Integer	—	Number of pulses
`tau_0`	Float	seconds	Base timing interval
`total_duration`	Float	seconds	Optional: total span
`pulse_width`	Float	seconds	Envelope FWHM
`interference_threshold`	Float	—	Maximum R allowed

4.4 Outputs

Output	Type	Description
`pulse_times`	List[Float]	Sequence of times $t_1, \ldots, t_n$
`interference_ratio`	Float	Computed R for sequence
`equal_baseline_ratio`	Float	R for equal-spaced comparison
`improvement_factor`	Float	R_{eq}/R_φ

4.5 Algorithms

4.5.1 Fibonacci Recursion for Efficiency

Exploit $\varphi^{k+1} = \varphi^k + \varphi^{k-1}$:

```
def phiPowers (n : Nat) : List Real :=
  let rec go (k : Nat) (prev curr : Real) : List Real :=
    if k = 0 then []
    else curr :: go (k-1) curr (prev + curr)
  go n 1 phi
```

4.5.2 Interference Computation

For Gaussian envelopes with width σ:

$$I_{\text{total}} = \sum_{i \neq j} \sqrt{\pi}\sigma \cdot e^{-(t_i - t_j)^2/4\sigma^2} \tag{5}$$

4.6 Performance Requirements

Metric	Requirement
Sequence generation time	< 10 ms for $n \leq 1000$
Timing precision	< 1 ps resolution
Export latency	< 100 ms to hardware

4.7 Verification

1. **Ratio bound:** $R_\varphi < \rho$ for all generated sequences
2. **Improvement:** $R_\varphi < R_{\text{eq}}$ for $n \geq 3$
3. **Timing accuracy:** $|t_k - \tau_0 \varphi^{k-1}| < 1$ fs

5 Module 3: Symmetry Verifier

5.1 Purpose

Provide real-time mathematical certification that proposed control adjustments will improve implosion symmetry, based on the Local Descent Link theorem.

5.2 Functional Requirements

Requirement 11 (SV-001: Ledger Computation)**.** *The system shall compute the Symmetry Ledger:*

$$J(r) = \sum_{i=1}^{n} w_i (r_i - 1)^2 \tag{6}$$

from intensity ratio measurements $r = (r_1, \ldots, r_n)$*.*

Requirement 12 (SV-002: Descent Verification)**.** *For any proposed adjustment yielding predicted ratios* r'*, the system shall verify:*

$$J(r') < J(r) \tag{7}$$

and return PASS or FAIL.

Requirement 13 (SV-003: Trust Region Check)**.** *The system shall verify that the current configuration is within the trust region:*

$$|r_i - 1| \leq \rho \quad \forall i \tag{8}$$

where ρ *is the certified operating envelope.*

Requirement 14 (SV-004: Certificate Generation)**.** *On PASS, the system shall generate a machine-checkable certificate containing:*

- *Current ledger value* $J(r)$
- *Predicted ledger value* $J(r')$
- *Descent margin* $J(r) - J(r')$
- *Timestamp and configuration ID*

Requirement 15 (SV-005: Audit Logging). *All verification decisions shall be logged with:*

- *Input measurements*
- *Proposed adjustment*
- *Decision (PASS/FAIL)*
- *Certificate (if PASS)*

5.3 Inputs

Parameter	Type	Units	Description
`intensity_ratios`	Array[Float]	—	Measured r_i values
`weights`	Array[Float]	—	Importance weights w_i
`proposed_adjustment`	Array[Float]	—	Control change Δu
`system_model`	Model	—	Predicts r' from Δu
`trust_radius`	Float	—	Operating envelope ρ

5.4 Outputs

Output	Type	Description
`decision`	Enum{PASS, FAIL}	Verification result
`current_ledger`	Float	$J(r)$
`predicted_ledger`	Float	$J(r')$
`descent_margin`	Float	$J(r) - J(r')$
`certificate`	Certificate	Machine-checkable proof
`trust_region_status`	Boolean	Within envelope?

5.5 Algorithms

5.5.1 Ledger Computation

```
def symmetryLedger (r : Fin n -> Real) (w : Fin n -> Real) : Real :=
  Finset.sum Finset.univ (fun i => w i * (r i - 1)^2)
```

5.5.2 Descent Check

```
def verifyDescent (J_current J_predicted : Real) : Bool :=
  J_predicted < J_current
```

5.6 Safety Requirements

Requirement 16 (SV-SAFE-001: Fail-Safe Default). *If verification cannot be completed (timeout, error), the system shall return FAIL.*

Requirement 17 (SV-SAFE-002: Envelope Violation Alarm). *If trust region is violated, the system shall:*

1. *Return FAIL*

2. *Trigger operator alarm*
3. *Log critical event*

Requirement 18 (SV-SAFE-003: Watchdog Timer)**.** *Verification shall complete within 10 ms or timeout with FAIL.*

5.7 Performance Requirements

Metric	**Requirement**
Verification latency	<1 ms typical, <10 ms maximum
Certificate generation	<5 ms
Availability	99.99% uptime

5.8 Verification

1. **Descent guarantee:** PASS implies $J(r') < J(r)$
2. **Soundness:** Never PASS when $J(r') \geq J(r)$
3. **Timeout safety:** All timeouts result in FAIL

6 Module 4: Jitter Simulator

6.1 Purpose

Quantify timing jitter tolerance budgets for laser hardware, enabling cost-effective component selection based on the Quadratic Advantage theorem.

6.2 Functional Requirements

Requirement 19 (JS-001: Degradation Model)**.** *The system shall compute expected performance degradation under jitter:*

$$D_{equal}(j) = \gamma j + O(j^2) \tag{9}$$

$$D_{\varphi}(j) = \beta j^2 + O(j^3) \tag{10}$$

for both equal-spaced and φ-spaced sequences.

Requirement 20 (JS-002: Tolerance Computation)**.** *Given maximum acceptable degradation $D_{\max}$, compute:*

$$j^{eq}_{\max} = D_{\max}/\gamma \tag{11}$$

$$j^{\varphi}_{\max} = \sqrt{D_{\max}/\beta} \tag{12}$$

Requirement 21 (JS-003: Monte Carlo Simulation)**.** *The system shall provide Monte Carlo simulation:*

- *Sample jitter from specified distribution*
- *Compute interference ratio for each sample*

- *Report statistics (mean, std, percentiles)*

Requirement 22 (JS-004: Hardware Specification). *Given jitter tolerance $j_{\max}$, output:*

- *Required oscillator stability*
- *Acceptable timing board specifications*
- *Cost comparison for component options*

Requirement 23 (JS-005: Comparison Report). *Generate report comparing:*

- *φ-scheduling vs equal-spacing tolerance*
- *Cost savings from relaxed jitter requirements*
- *Risk assessment for different hardware choices*

6.3 Inputs

Parameter	Type	Units	Description
`pulse_sequence`	List[Float]	seconds	Nominal pulse times
`jitter_distribution`	Distribution	—	Gaussian, uniform, etc.
`jitter_magnitude`	Float	seconds	Standard deviation j
`max_degradation`	Float	—	Acceptable $D_{\max}$
`n_samples`	Integer	—	Monte Carlo sample count

6.4 Outputs

Output	Type	Description
`degradation_mean`	Float	Expected $D(j)$
`degradation_std`	Float	Standard deviation
`max_jitter_equal`	Float	$j_{\max}^{\text{eq}}$
`max_jitter_phi`	Float	$j_{\max}^{\varphi}$
`improvement_ratio`	Float	$j_{\max}^{\varphi}/j_{\max}^{\text{eq}}$
`hardware_recommendations`	Report	Component specifications

6.5 Algorithms

6.5.1 Analytical Degradation

```
def degradationFormula (isPhiSpaced : Bool) (j : Real) : Real :=
  if isPhiSpaced then
    beta * j^2  -- Quadratic
  else
    gamma * j   -- Linear
```

6.5.2 Monte Carlo Engine

1. Generate N jitter samples from distribution
2. For each sample, perturb pulse times
3. Compute interference ratio
4. Aggregate statistics

6.6 Performance Requirements

Metric	Requirement
Analytical computation	< 100 ms
Monte Carlo (10^6 samples)	< 10 seconds
Report generation	< 1 second

6.7 Verification

1. **Quadratic bound:** $D_\varphi(j) \leq \beta j^2$ for $j \leq j_{\max}$
2. **Monte Carlo accuracy:** Mean within 5% of analytical
3. **Improvement ratio:** $j_{\max}^\varphi / j_{\max}^{\text{eq}} \geq 3$ typically

7 Integration Requirements

7.1 Module Interoperability

Requirement 24 (INT-001: Data Format Consistency)**.** *All modules shall use consistent data formats:*

- *Time values: 64-bit IEEE 754 floating point, SI seconds*
- *Nuclear configurations: (Z, N) integer pairs*
- *Ratios: Dimensionless floating point*

Requirement 25 (INT-002: API Compatibility)**.** *Modules shall expose:*

- *Python bindings for scripting*
- *C/C++ interface for real-time systems*
- *REST API for distributed operation*

Requirement 26 (INT-003: Logging Standard)**.** *All modules shall log to common format with:*

- *ISO 8601 timestamps*
- *Severity levels (DEBUG, INFO, WARN, ERROR, CRITICAL)*
- *Structured JSON payloads*

7.2 Deployment Options

Deployment	Modules	Use Case
Workstation	All	Development, analysis
Control room	PSE, SV	Shot operations
FPGA	SV core	Real-time control
Cloud	RNO, JS	Design studies

8 Formal Verification Traceability

8.1 Lean 4 Proof Artifacts

Each module's critical algorithms are backed by formal proofs:

Module	Proof File
Reaction Network Optimizer	`Fusion/ReactionNetwork.lean`
φ-Scheduler Engine	`Fusion/InterferenceBound.lean`
Symmetry Verifier	`Fusion/LocalDescent.lean`
Jitter Simulator	`Fusion/JitterRobustness.lean`

8.2 Theorem-to-Requirement Mapping

Requirement	Theorem	Status
RNO-002	`stabilityDistance_zero_of_doublyMagic`	Verified
RNO-004	`doublyMagic_is_minimum`	Verified
PSE-002	`phi_interference_bound_exists`	Verified
SV-002	`local_descent_link`	Verified
JS-001	`phi_scheduling_quadratic`	Verified

9 Appendix: Mathematical Definitions

9.1 Magic Numbers

$$\mathcal{M} = \{2, 8, 20, 28, 50, 82, 126\} \tag{13}$$

9.2 Distance to Magic

$$d(x) = \min_{m \in \mathcal{M}} |x - m| \tag{14}$$

9.3 Stability Distance

$$S(Z, N) = d(Z) + d(N) \tag{15}$$

9.4 Golden Ratio

$$\varphi = \frac{1 + \sqrt{5}}{2} \approx 1.6180339887... \tag{16}$$

9.5 Symmetry Ledger

$$J(r) = \sum_{i=1}^{n} w_i (r_i - 1)^2 \tag{17}$$

9.6 Interference Ratio

$$R = \frac{\sum_{i \neq j} C_{ij}}{n \int E(t)^2 dt} \tag{18}$$

9.7 Degradation Functions

$$D_\varphi(j) = \beta j^2 + O(j^3) \tag{19}$$

$$D_{\mathrm{eq}}(j) = \gamma j + O(j^2) \tag{20}$$