AXILOG

SRS-005-CR-001

Conformance & Remediation

Locked v1.1 L2 [D1]

Last updated 27 March 2026
Supersedes SRS-005-CR-001 v1.0
Depends on SRS-005
Changelog 1 entry

Contents

Depends on: SRS-005 v1.1-Frozen **Scope:** Arithmetic Conformance

---

0. GOVERNING PRINCIPLE

> **All arithmetic in L2 MUST conform to SRS-005.**
>
> **L2 SHALL NOT define independent arithmetic semantics.**
>
> **L2 SHALL be a consumer of L1 arithmetic law, not a variant of it.**

Any divergence from SRS-005 is a system integrity failure.

---

1. PURPOSE

This document defines the mandatory remediation required to bring `certifiable-inference` into full conformance with SRS-005 v1.1-Frozen.

It specifies:

- Identified non-conformances
- Required code-level corrections
- Deferred compliance items
- Golden reference impact
- Verification requirements

This document does not redefine arithmetic.

It enforces compliance.

---

2. CONFORMANCE REQUIREMENT

A `certifiable-inference` implementation is conformant only if:

- All blocking and high-severity issues are remediated
- No SRS-005-SHALL violations remain in arithmetic paths
- Golden vectors are regenerated where semantics changed
- Cross-platform identity is preserved post-remediation
- Type definitions are sourced from L1 (no duplication)

---

3. NON-CONFORMANCE REGISTER

CR-001-ISSUE-001 — Signed Left Shift (UB)

Field	Value
Component	`fixed_from_int`
Violation	SRS-005-SHALL-068
Severity	🔴 BLOCKING

Description

Left-shift of signed integer invokes undefined behaviour for negative values in C99.

```
return (fixed_t)(-((-i) << FIXED_SHIFT));
```

Why This Is Dangerous

C99 §6.5.7: "If E1 has a signed type and nonnegative value, and E1 × 2^E2 is representable in the result type, then that is the resulting value; otherwise, the behavior is undefined."

For negative values, left-shift is undefined — not implementation-defined, but undefined. The compiler may do anything.

CR-001-ISSUE-002 — Signed Right Shift (Implementation-Defined)

Field	Value
Component	`fixed_to_int`
Violation	SRS-005-SHALL-068
Severity	🔴 BLOCKING

Description

Right-shift of signed integer is implementation-defined and violates cross-platform determinism.

```
return f >> FIXED_SHIFT;
```

Why This Is Dangerous

C99 §6.5.7: “If E1 has a signed type and a negative value, the resulting value is implementation-defined.”

While most compilers perform arithmetic shift (sign-extension), the standard permits logical shift (zero-fill). This breaks bit-identity across platforms.

CR-001-ISSUE-003 — Multiplication Rounding Divergence

Field	Value
Component	`fixed_mul`
Violation	SRS-005-SHALL-067
Severity	🔴 BLOCKING

Description

Use of rounding (`+ FIXED_HALF`) violates mandated truncation semantics.

```
result += FIXED_HALF;
return (fixed_t)(result >> FIXED_SHIFT);
```

Why This Is Dangerous

SRS-005-SHALL-067 mandates truncation toward zero. The previous implementation was “better math” (rounding) but “worse for a verifiable substrate” — rounding introduces hidden dependencies on tie-handling behaviour. Truncation is simple, total, and identical everywhere.

CR-001-ISSUE-004 — Signed Left Shift in Division

Field	Value
Component	`fixed_div`
Violation	SRS-005-SHALL-068
Severity	🔴 BLOCKING

Description

Left-shift used for scaling introduces undefined behaviour.

```
int64_t numerator = (int64_t)a << FIXED_SHIFT;
```

Remediation

Use multiplication instead of shift:

```
int64_t numerator = (int64_t)a * FIXED_ONE;
```

CR-001-ISSUE-005 — INT32_MIN Overflow (abs/neg)

Field	Value
Component	`fixed_abs`, `fixed_neg`
Violation	SRS-005-SHALL-014, SRS-005-SHALL-015
Severity	🟠 HIGH

Description

Negation of `INT32_MIN` overflows and is not handled.

```
-INT32_MIN  /* Undefined behaviour: result cannot be represented */
```

Required End State

- Saturate to `INT32_MAX`
- Set `overflow` flag (when fault propagation is integrated)

CR-001-ISSUE-006 — Duplicate Type Definition (L2 Header Leak)

Field	Value
Component	`include/fixed_point.h`
Violation	Architectural — Source of Truth
Severity	🟠 HIGH

Description

Added in v1.1 per external audit finding.

`certifiable-inference` defines its own `fixed_point.h` with `FIXED_ONE`, `FIXED_HALF`, etc. This creates **two sources of truth** for the same constants:

- `libaxilog/include/axilog/types.h` (L1 — authoritative)
- `certifiable-inference/include/fixed_point.h` (L2 — duplicate)

Risk

If L1 constants ever change (e.g., for a Q16.32 variant), the L2 definitions would silently diverge, breaking the "one arithmetic law" principle.

Remediation

Replace the contents of L2's `fixed_point.h` with:

```
/**
 * @file fixed_point.h
 * @brief L2 Fixed-Point Types (L1 Delegation)
 *
 * DVEC: v1.3
 * DETERMINISM: D1 — Strict Deterministic
 * SRS: SRS-005-CR-001
 *
 * This header delegates to L1 (libaxilog) for all arithmetic
 * type definitions. L2 SHALL NOT define independent constants.
 */

#ifndef CERTIFIABLE_INFERENCE_FIXED_POINT_H
#define CERTIFIABLE_INFERENCE_FIXED_POINT_H

#include <axilog/types.h>

/* L2 legacy type aliases — delegate to L1 */
typedef q16_16_t fixed_t;

#define FIXED_SHIFT   Q16_SHIFT
#define FIXED_ONE     Q16_ONE
#define FIXED_HALF    Q16_HALF
#define FIXED_MAX     Q16_MAX
#define FIXED_MIN     Q16_MIN
#define FIXED_ZERO    0

#endif /* CERTIFIABLE_INFERENCE_FIXED_POINT_H */
```

Rationale

This ensures the entire stack updates atomically if L1 constants change. L2 becomes a pure consumer of L1 arithmetic law.

---

4. REQUIRED REMEDIATION

4.1 `fixed_from_int`

```
/**
 * @brief Convert integer to Q16.16 fixed-point.
 *
 * SRS-005-SHALL-005: Integer-to-Q16.16 conversion
 * SRS-005-SHALL-068: No signed shift
 */
static inline fixed_t fixed_from_int(int32_t i)
{
    return (fixed_t)(i * FIXED_ONE);
}
```

4.2 `fixed_to_int`

```
/**
 * @brief Convert Q16.16 fixed-point to integer (truncation).
 *
 * SRS-005-SHALL-006: Q16.16-to-integer conversion
 * SRS-005-SHALL-068: No signed shift
 */
static inline int32_t fixed_to_int(fixed_t f)
{
    return f / FIXED_ONE;
}
```

4.3 `fixed_mul`

```
/**
 * @brief Saturated Q16.16 multiplication (truncation toward zero).
 *
 * SRS-005-SHALL-012: Saturated multiplication
 * SRS-005-SHALL-067: Truncation mandated (no rounding)
 * SRS-005-SHALL-068: No signed shift
 */
fixed_t fixed_mul(fixed_t a, fixed_t b)
{
    int64_t result = (int64_t)a * (int64_t)b;

    /* Truncation toward zero — C99 division semantics */
    result = result / FIXED_ONE;

    /* Saturation */
    if (result > INT32_MAX) {
        return INT32_MAX;
    }
    if (result < INT32_MIN) {
        return INT32_MIN;
    }

    return (fixed_t)result;
}
```

4.4 `fixed_div`

```
/**
 * @brief Saturated Q16.16 division.
 *
 * SRS-005-SHALL-013: Saturated division
 * SRS-005-SHALL-068: No signed shift
 * SRS-005-SHALL-069: INT32_MIN/-1 handled
 */
fixed_t fixed_div(fixed_t a, fixed_t b)
{
    if (b == 0) {
        return FIXED_ZERO;  /* Temporary — see §5 (CR-001-DEFER-003) */
    }

    /* Use multiplication instead of shift */
    int64_t numerator = (int64_t)a * FIXED_ONE;
    int64_t result = numerator / (int64_t)b;

    /* Saturation */
    if (result > INT32_MAX) {
        return INT32_MAX;
    }
    if (result < INT32_MIN) {
        return INT32_MIN;
    }

    return (fixed_t)result;
}
```

4.5 `fixed_abs`

```
/**
 * @brief Absolute value with INT32_MIN saturation.
 *
 * SRS-005-SHALL-015: Absolute value
 * SRS-005-SHALL-069: INT32_MIN overflow handled
 */
fixed_t fixed_abs(fixed_t f)
{
    if (f == INT32_MIN) {
        return INT32_MAX;  /* Saturate — see §5 for fault integration */
    }
    return (f < 0) ? -f : f;
}
```

4.6 `fixed_neg`

```
/**
 * @brief Negation with INT32_MIN saturation.
 *
 * SRS-005-SHALL-014: Negation
 * SRS-005-SHALL-069: INT32_MIN overflow handled
 */
fixed_t fixed_neg(fixed_t f)
{
    if (f == INT32_MIN) {
        return INT32_MAX;  /* Saturate — see §5 for fault integration */
    }
    return -f;
}
```

---

5. DEFERRED COMPLIANCE ITEMS

CR-001-DEFER-001 — Fault Propagation Integration

Field	Value
Area	All arithmetic primitives
Target	Phase 3

Description

Current implementation does not propagate `ct_fault_flags_t`.

Required End State

All functions SHALL conform to:

- SRS-005-SHALL-029 (fault context acceptance)
- SRS-005-SHALL-031 (fault signalling)
- SRS-005-SHALL-034 (no silent failure)

Migration Path

```
/* Current (Phase 2) */
fixed_t fixed_mul(fixed_t a, fixed_t b);

/* Target (Phase 3) */
fixed_t fixed_mul(fixed_t a, fixed_t b, ct_fault_flags_t *faults);
```

CR-001-DEFER-002 — INT32_MIN Fault Signalling

Field	Value
Area	`fixed_abs`, `fixed_neg`
Target	Phase 3

Current Behaviour

Saturates to `INT32_MAX` silently.

Required Behaviour

```
if (f == INT32_MIN) {
    faults->overflow = 1;
    return INT32_MAX;
}
```

CR-001-DEFER-003 — Division by Zero Fault

Field	Value
Area	`fixed_div`
Target	Phase 3

Current Behaviour

```
return FIXED_ZERO;
```

Required Behaviour

```
faults->div_zero = 1;
return 0;
```

---

6. GOLDEN REFERENCE IMPACT

6.1 Affected Operations

Operation	Change Type	Impact
`fixed_mul`	**MANDATORY**	Rounding → Truncation
Inference outputs	**CASCADE**	All dependent computations

6.2 Required Actions

1. Apply all §4 remediations
2. Re-run `certifiable-harness`
3. Regenerate golden vectors
4. Freeze new reference set
5. Update cross-platform identity hashes

6.3 Interpretation

> **Changes are corrections, not regressions.**
>
> **Previous outputs are non-conformant to SRS-005.**

The new golden references represent the first **correct** outputs. Prior references were produced by non-conformant arithmetic.

---

7. VERIFICATION REQUIREMENTS

The following MUST pass post-remediation:

```
□ property-test-arith        — All arithmetic property tests
□ golden-vector              — Re-frozen reference vectors
□ cross-platform-x86         — x86_64 bit-identity
□ cross-platform-arm         — ARM64 bit-identity
□ static-analysis            — No UB (clang-tidy, cppcheck)
□ forbidden-pattern-scan     — No << or >> on signed ints
□ rtm-coverage               — SRS-005 alignment verified
□ header-delegation-check    — L2 types delegate to L1
```

---

8. NON-SCOPE CONFIRMATION

The following components are explicitly **out of scope**:

8.1 Jenkins Hash

The Jenkins One-at-a-Time hash is:

- **Deterministic** — integer-only, bit-perfect
- **Platform-independent** — no signed shifts
- **Internal scope** — hash table lookup, not cryptographic
- **Not part of AX:OBS:v1** — no Chain of Custody role

Status: DO NOT MODIFY

Rationale: Changing it would introduce unnecessary "Golden Vector noise" without increasing security or determinism.

---

9. ARCHITECTURAL DIRECTION

9.1 Progressive L1 Adoption (Recommended)

Future alignment path:

```
/* Preferred (L1 primitive) */
q16_16_t result = ax_mul_q16(a, b, &faults);

/* Transitional (L2 wrapper) */
fixed_t result = fixed_mul(a, b);
```

9.2 Header Consolidation (Mandatory)

Per CR-001-ISSUE-006, L2 SHALL delegate type definitions to L1:

```
certifiable-inference/include/fixed_point.h
    └──── #include <axilog/types.h>
        └──── typedef q16_16_t fixed_t;
```

9.3 Rationale

- Eliminates duplication
- Prevents future drift

- Reduces audit surface
- Ensures atomic updates across the stack

---

10. BUILD SYSTEM INTEGRATION

10.1 CMakeLists.txt Updates

```
# Link against libaxilog
find_package(axilog REQUIRED)
target_link_libraries(certifiable-inference PRIVATE axilog::axilog)

# Include L1 headers
target_include_directories(certifiable-inference PRIVATE
    ${AXILOG_INCLUDE_DIRS}
)

# Forbidden pattern scan (CI gate)
add_custom_target(forbidden-pattern-scan
    COMMAND grep -rn "<<\\|>>" --include="*.c" --include="*.h"
            ${CMAKE_SOURCE_DIR}/src ${CMAKE_SOURCE_DIR}/include
            && exit 1 || exit 0
    COMMENT "Scanning for forbidden shift operators on signed integers"
)
```

10.2 Hardened Compiler Flags

```
add_compile_options(
    -Wall -Wextra -Werror -Wpedantic
    -fno-strict-aliasing
    -fwrapv
    -fno-tree-vectorize
    -fno-builtin
    -ffp-contract=off
    -fno-fast-math
)
```

---

11. FINAL STATEMENT

> **`certifiable-inference` SHALL NOT define arithmetic behaviour.**
>
> **It SHALL implement SRS-005.**

Following remediation:

- ✅ All undefined behaviour is eliminated
- ✅ All implementation-defined behaviour is removed
- ✅ Arithmetic semantics are aligned with L1
- ✅ Cross-platform determinism is preserved
- ✅ Type definitions delegate to single source of truth

---

12. CLOSURE CONDITION

This document is satisfied when:

- ☐ All BLOCKING issues are remediated (001–004, 006)
- ☐ All HIGH issues are remediated (005)
- ☐ Golden vectors are re-frozen
- ☐ CI gates pass
- ☐ No SRS-005 violations remain in arithmetic paths
- ☐ L2 headers delegate to L1 types

---

13. REVISION HISTORY

Version	Date	Author	Changes
1.0-Frozen	2026-03-27	William Murray	Initial conformance specification — 5 issues, full remediation defined
1.1-Frozen	2026-03-27	William Murray	Added CR-001-ISSUE-006 (L2 header leak), build system integration, hardened compiler flags. Total: 6 issues.

14. AUDIT RECORD

v1.0 → v1.1

Finding	Source	Resolution
Duplicate type definition in L2 header	Gemini 2.5 Pro	CR-001-ISSUE-006 added
Build system integration missing	Internal	§10 added
Hardened compiler flags not specified	Internal	§10.2 added

Audit Source: External review (Gemini 2.5 Pro)

Audit Date: 27 March 2026

15. AUDIT VERDICT

Category	Status
Blocking issues	✅ Remediation defined (4)
High issues	✅ Remediation defined (2)
Arithmetic alignment	✅ Achieved post-patch
Header consolidation	✅ Specified
Determinism compliance	✅ Enforced
Remaining risk	🟡 Deferred (fault propagation — Phase 3)

Appendix A — Shift Operator Audit Checklist

Pre-merge verification:

```
# Must return empty (no matches)
grep -rn '<<\|>>' --include='*.c' --include='*.h' \
    src/ include/ | grep -v '//'

# If any matches found, verify they are:
# 1. On unsigned types only, OR
# 2. Not in arithmetic paths (e.g., bit flags)
```

Appendix B — Cross-Reference Matrix

L2 Issue	SRS-005 SHALL	Remediation
CR-001-ISSUE-001	SHALL-068	§4.1
CR-001-ISSUE-002	SHALL-068	§4.2
CR-001-ISSUE-003	SHALL-067	§4.3
CR-001-ISSUE-004	SHALL-068	§4.4
CR-001-ISSUE-005	SHALL-014, 015	§4.5, §4.6
CR-001-ISSUE-006	(Architectural)	§9.2

SRS-005-CR-001 v1.1-Frozen — Audit-Frozen FINAL William Murray · SpeyTech · March 2026 Patent GB2521625.0