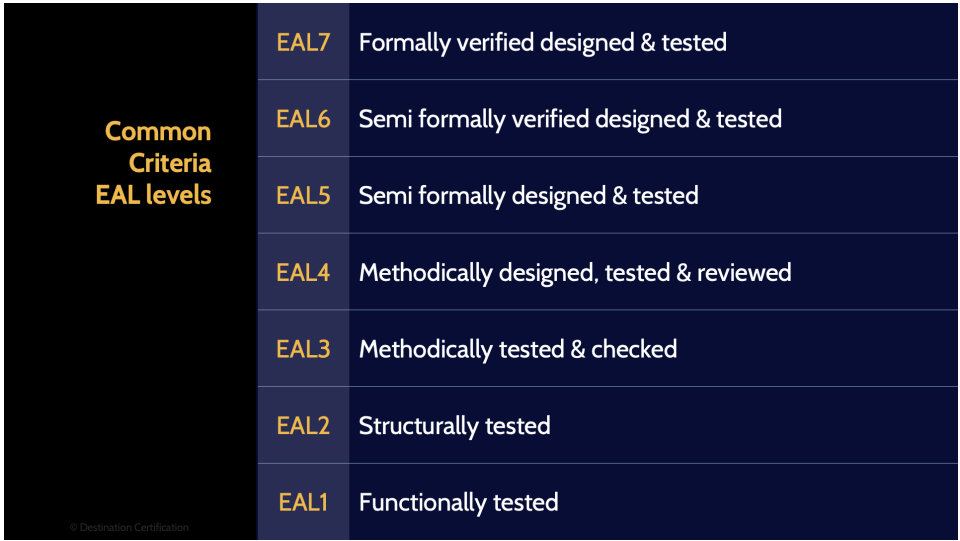
The GNU libc atanh is correctly rounded

Paul Zimmermann

33rd IEEE International Symposium on Computer Arithmetic, June 30th, 2026

Evaluation Assurance Level



Where are we for mathematical libraries?

- EAL 0: no specification

IEEE 754 $\leq$ 2018

Where are we for mathematical libraries?

- EAL 3: correct rounding — RLIBM, LLVM libc, CORE-MATH, IEEE 754-2029?
- EAL 0: no specification — IEEE 754 $\leq$ 2018

Where are we for mathematical libraries?

- EAL 5: pen-and-paper proof — CORE-MATH-PROOF
- EAL 3: correct rounding — RLIBM, LLVM libc, CORE-MATH, IEEE 754-2029?
- EAL 0: no specification — IEEE 754 $\leq$ 2018

Where are we for mathematical libraries?

- EAL 7: formally verified designed & tested — IEEE 754-2099?
- EAL 5: pen-and-paper proof — CORE-MATH-PROOF
- EAL 3: correct rounding — RLIBM, LLVM libc, CORE-MATH, IEEE 754-2029?
- EAL 0: no specification — IEEE 754 $\leq$ 2018

The GNU libc

Main mathematical library under Linux.

Highly portable (aarch64, x86, arm, hppa, hurd, i386, m68k, mips, powerpc, riscv, s390, sparc, x86, x86-64).

Very active development. New release every 6 months.

Since release 2.41 (January 2025), provides some correctly-rounded functions (from CORE-MATH project).

Since release 2.42 (July 2025), provides all C23 mathematical functions (only mathematical library to provide all of them).

Correct rounding in GNU libc

Latest release is 2.43 (January 2026).

Thirty binary32 functions are CR: acos, acosh, asin, asinh, atan, atanh, cbrt, cosh, erf, erfc, expm1, lgamma, log10, log1p, sinh, tan, tanh, tgamma, acospi, asinpi, atanpi, cospi, exp10m1, exp2m1, log10p1, log2p1, sinpi, sqrt, tanpi, atan2, atan2pi.

[Still non CR binary32 functions: cos, exp, exp2, exp10, log, log2, sin, rsqrt, hypot, pow.]

Seven binary64 functions are (claimed to be) CR: acosh, asinh, **atanh**, erf, erfc, lgamma, tgamma.

This talk

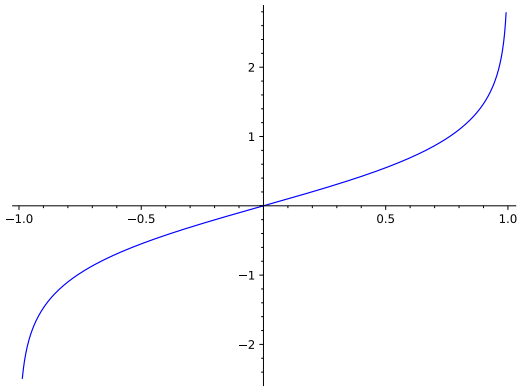
New project started in 2026: prove all CORE-MATH binary64 functions.

Since atanh was being integrated into GNU libc, try to prove this function.

Main difficulty: not possible to change the code to make the proof easier (GNU libc 2.43 freeze).

The atanh function

$$\operatorname{atanh}(x) = \frac{1}{2} \log \frac{1+x}{1-x}$$



Origin of the code

Original code written by Alexei Sibidanov for CORE-MATH. Integrated into GNU libc by Adhemerval Zanella, and reviewed for GNU libc by DJ Delorie.

```
The CORE-MATH implementation is correctly rounded for any rounding mode.
The code was adapted to glibc style and to use the definition of
math_config.h (to handle errno, overflow, and underflow).

Benchtest on x64_64 (Ryzen 9 5900X, gcc 14.2.1), aarch64 (Neoverse-N1,
gcc 13.3.1), and powerpc (POWER10, gcc 13.2.1) shows:
```

reciprocal-throughput	master	patched	improvement
x86_64	26.4969	22.4625	15.23%
x86_64v2	26.0792	22.9822	11.88%
x86_64v3	25.6357	22.2147	13.34%
aarch64	20.2295	19.7001	2.62%
power10	10.0986	9.3846	7.07%

Branches for atanh

- check for special values: NaN, $\pm\infty$, $|x| \geq 1$
- branch $0 \leq x < x_0 := \texttt{0x1.d12ed0af1a27fp-27}$
- branch $x_0 \leq x < 1/4$
- branch $1/4 \leq x < 1$

Since $\mathrm{atanh}(-x) = -\mathrm{atanh}(x)$, and the approximations are odd too, it might be enough to consider $x \geq 0$ in the proofs.

Consider $p(x) = x \cdot q(x)$ where $q(x)$ is even.
For RU, $\tilde{q}(x) = \tilde{q}(-x)$ since $\mathrm{RU}(x^2) = \mathrm{RU}((-x)^2)$.
However, $\mathrm{RU}(x \cdot \tilde{q}(x)) \neq -\mathrm{RU}((-x) \cdot \tilde{q}(-x))$.

Branch $0 \le x < x_0$

```
return fma (x, 0x1p-55, x);
```

Lemma

For $0 \le x < x_0$, $\mathrm{fma}(x, 2^{-55}, x)$ yields the correct rounding of $\mathrm{atanh}(x)$, whatever the rounding mode.

Proof sketch: ok for $x = \pm 0$.

For $0 < x < x_1 \approx 1.29 \cdot 10^{-8}$, $x < \mathrm{atanh}(x) < x + 2^{-54}x$, and there is no rounding boundary in $(x, x + 2^{-54}x)$.

For $x_1 \le x < x_0$, $\mathrm{ulp}(x) = 2^{-79}$, and $x < \mathrm{atanh}(x) < \frac{1}{2}\mathrm{ulp}(x)$.

Details in the proceedings.

Branch $1/4 \le x < 1$

Checked by brute-force search.

Idea: on small intervals $x + ju$, with $u = \mathrm{ulp}(x)$, $0 \le j < n$, compare the computed result y to the approximation $z = \circ(a + jb)$, where $a \approx \mathrm{atanh}(x)$, $b \approx u \cdot \mathrm{atanh}'(x)$ are computed with larger precision using GNU MPFR.

If say $n = 10^4$, the amortized cost of z is a few cycles only.

Two binades to check, where each binade has 2^{52} numbers.

Check all four rounding modes (to nearest, toward zero, toward $\pm\infty$).

Branch $1/4 \leq x < 1$

We also have to check with or without FMA contraction:

```
double f = dx2*((c[0] + dx*c[1]) + dx2*(c[2] + dx*c[3] + dx2*c[4]));
```

`c[0] + dx*c[1]` might be evaluated as $\circ(c_0 + \circ(d_x \cdot c_1))$ without FMA, or $\circ(c_0 + d_x \cdot c_1)$ with FMA.

Total 2^{56} calls to check. Each call takes about 60 cycles on average. On a 3Ghz machine, this is about 46 core-years.

Brute-force method independently used by Toru Koizumi et al. to check correct rounding for the binary64 exponential function (IPSJ SIG Tech. Rep., 2023-HPC-192, in Japanese).

Branch $x_0 \le x < 1/4$

Two-level strategy (Ziv's strategy):

- the fast path evaluates $\mathrm{atanh}(x)$ with a few guard bits, while ensuring a rigorous error bound:
$$y - \epsilon \le \mathrm{atanh}(x) \le y + \epsilon.$$
If $y - \epsilon$ and $y + \epsilon$ round to the same value z (rounding test) then $\mathrm{atanh}(x)$ rounds to z too.
- the accurate path evaluates $\mathrm{atanh}(x)$ with a larger precision, enough to guarantee correct rounding, except maybe for the hard-to-round cases.

It then suffices to check the hard-to-round cases are correctly rounded.

Hard-to-round cases for atanh

Computed with BaCSeL software tool, using the SLZ algorithm.

Found 27077 inputs $x \in [x_0, 1)$ such that $\mathrm{atanh}(x)$ has ≥ 43 identical bits after the round bit.

Among those, 25089 inputs in $[x_0, 1/4)$.

The largest number of identical bits after the round bit is 72:

$$\mathrm{atanh}(\texttt{0x1.dfffffffffabap-21}) =$$
$$2^{-21} \cdot \underbrace{1.11100000000000000000000000000000000000000001110000011}_{53}\,1\,\underbrace{111\ldots111}_{72}0100\ldots$$

Hardest-to-round case in $[2^{-17}, 1/4)$:

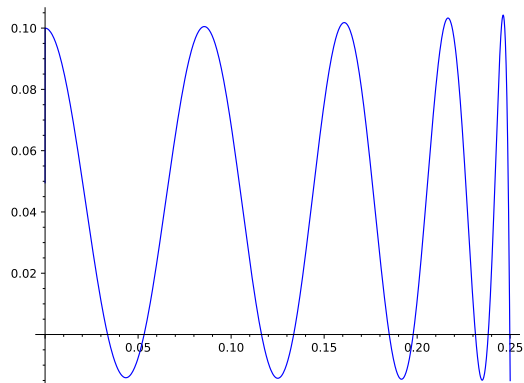
$$\mathrm{atanh}(\texttt{0x1.447027c857b99p-12}) =$$
$$2^{-12} \cdot \underbrace{1.01000100011100000010100001110110000010100101111110010}_{53}\,0\,\underbrace{111\ldots111}_{55}0111\ldots$$

Fast path for $x_0 \le x < 1/4$

Approximate $\operatorname{atanh}(x)$ with a minimax polynomial:

$$p(x) = x + (h + \ell)x^3 + c_0x^5 + c_1x^7 + \cdots + c_8x^{21}.$$

Graph of $2^{53}(p(x) - \operatorname{atanh}(x))/x^5$ for $x_0 \le x < 1/4$:



Fast path for $x_0 \leq x < 1/4$

$$p(x) = x + (h + \ell)x^3 + c_0x^5 + c_1x^7 + \cdots + c_8x^{21}.$$

- compute a double-double approximation $x_3 + d_3 \approx x^3$
- approximate $q = c_0 + c_1x^2 + \cdots + c_8x^{16}$ in binary64.
- approximate $t = x^2q + \ell$ with an FMA
- approximate $h + t$ with a Fast2Sum $\rightarrow a_h + a_\ell$
- multiply by $x_3 + d_3 \rightarrow b_h + b_\ell$
- add x to b_h with a Fast2Sum $\rightarrow p_h + p_\ell$
- add b_ℓ to p_ℓ

Bounding the rounding error on t with Gappa

```
@rnd = float<ieee_64,RND>;
x2 = x*x;
x2r rnd= x*x;
...
t = x2*p+l;
tr = rnd(x2r*pr+l);
{ x in [XMIN,XMAX] -> |tr-t|/x2r in ? }
$ x in STEP;
```

Gappa proves:

$$|\mathtt{tr} - (\ell + c_0 x^2 + c_1 x^4 + \cdots + c_8 x^{18})| < 2^{-52.3079} \cdot \mathtt{x2r}$$

Bounding the rounding error

With additional pen-and-paper analysis:

$$|p_h + p_\ell - p(x)| < 2^{-103.967}x + 2^{-101.871}x^3 + 2^{-52.3078}x^5$$

Sollya proves:

$$|p(x) - \operatorname{atanh}(x)| < 2^{-56.261}x^5$$

We deduce:

$$|p_h + p_\ell - p(x)| < 2^{-103.967}x + 2^{-101.871}x^3 + 2^{-52.2175}x^5$$

Rounding test

$$\epsilon = \circ\left(x \cdot \circ\left(x_4 \cdot \frac{29}{2^{57}}\right) + 2^{-103}\right), \qquad \ell_b = p_h + \circ(p_\ell - \epsilon), \qquad u_b = p_h + \circ(p_\ell + \epsilon)$$

Theorem

Let x a binary64 number, $x_0 \le x < 1/4$. If $\circ(\ell_b) = \circ(u_b)$, $\circ(\ell_b)$ is the correct rounding of $\operatorname{atanh}(x)$.

Since $\ell_b = p_h + \circ(p_\ell - \epsilon)$, we have $\ell_b \le p_h + p_\ell$.

Main idea: prove that $p_h + p_\ell$ is closer from $\operatorname{atanh}(x)$ than from ℓ_b, thus:

$$\ell_b \le p_h + p_\ell \le \operatorname{atanh}(x) \qquad \text{or} \qquad \ell_b \le \operatorname{atanh}(x) \le p_h + p_\ell$$

It follows $\ell_b \le \operatorname{atanh}(x)$ thus $\circ(\ell_b) \le \circ(\operatorname{atanh}(x))$. See proceedings for details.

Accurate path for $x_0 \leq x < 1/4$

A degree-37 polynomial is used:

$$q(x) = x + c_3 x^3 + \cdots + c_{37} x^{37}$$

Coefficients of degree $3 - 27$ are double-double, upper coefficients are double.

Sollya bound:

$$|q(x) - \operatorname{atanh}(x)| < 2^{-111.784} \operatorname{atanh}(x)$$

The code involves operations with double-double coefficients (addition, multiplication), using classical algorithms (Fast2Sum, MulDD).

Gappa bound (750-line script, 3 minutes per rounding mode):

$$|y_0 + y_1 - q(x)| < 2^{-101.681} \operatorname{atanh}(x)$$

$$|y_0 + y_1 - \operatorname{atanh}(x)| < 2^{-101.679} \operatorname{atanh}(x)$$

Accurate path

Theorem

For any binary64 number x such that $\mathrm{atanh}(x)$ has less than 47 identical bits after the round bit, if $y_0 + y_1$ is the double-double approximation computed by the accurate path, then $\circ(y_0 + y_1)$ is the correct rounding of $\mathrm{atanh}(x)$.

Proof.

Let $y = \mathrm{atanh}(x)$. It follows y is at distance at least $2^{-48}\mathrm{ulp}(y)$ from any rounding boundary z: $|y - z| \geq 2^{-48}\mathrm{ulp}(y) \geq 2^{-101}|y|$ since $\mathrm{ulp}(y) > 2^{-53}|y|$. Thus $|y_0 + y_1 - y| < 2^{-101.679}|y|$ implies $|y_0 + y_1 - y| < |y - z|$, i.e., $y_0 + y_1$ is closer from $y = \mathrm{atanh}(x)$ than any rounding boundary. Thus $\circ(y_0 + y_1) = \circ(\mathrm{atanh}(x))$. □

Since CORE-MATH contains all hard-to-round cases with at least 43 identical bits after the round bit, it remains to check correct rounding for those.

Summary

- branch $0 \leq x < x_0$: pen-and-paper proof (no accurate path)
- branch $x_0 \leq x < 1/4$: proof with Sollya, Gappa, and pen-and-paper (both for fast path and accurate path)
- branch $1/4 \leq x < 1$: proof by exhaustive tests

This work would not have been possible without...

Alexei Sibidanov who implemented the atanh function in CORE-MATH.

Adhemerval Zanella who integrated this code into GNU libc.

DJ Delorie who reviewed the GNU libc integration.

The authors of Sollya, in particular Sylvain Chevillard, Christop Lauter and Mioara Joldeș.

The authors of Gappa, in particular Guillaume Melquiond.

Claude-Pierre Jeannerod, Wonyeol Lee and Hyunsoo Lee who made useful comments.

The three anonymous reviewers.

The Grid 5000 testbed which made exhaustive tests possible.

Limits of the pen-and-paper proof

The fast path looks like:

```
double lb = h + (l - eps), ub = h + (l + eps);
if (lb == ub) return lb; else return accurate (x);
```

Let $\ell_b = h + \circ(\ell - \epsilon)$ and $u_b = h + \circ(\ell + \epsilon)$ before final rounding. The proof first shows:

$$\ell_b \leq \mathrm{atanh}(x) \leq u_b \tag{1}$$

and deduces:

$$\circ(\ell_b) \leq \circ(\mathrm{atanh}(x)) \leq \circ(u_b)$$

But it can be that Eq. (1) fails with a smaller value of ϵ, and still $\circ(\ell_b) = \circ(\mathrm{atanh}(x))$, especially if $\mathrm{atanh}(x)$ is far from a rounding boundary.

Only the exhaustive search approach can find the smallest possible ϵ.

Conclusion and Future work

First proof of correctness for a binary64 function from a well-established library.

Full proofs already available for CORE-MATH `acos`, `acosh`, `asin`, `cosh`, `sinh`.

Work in progress for `asinh` and `tanh` (Cyprien Peignier).

Then prove the next functions to be integrated into GNU libc.

Convert these pen-and-paper proofs into formal proofs?

For functions defined over the full binary64 range, like $\log(x)$, the brute-force approach is not feasible with academic resources, and the general case (far from 0) is too complex for Gappa.

Maybe feasible with a SETI@home/BOINC approach?