

RTL

The coordinate primitive in RTL: decode, compose, and distance

Taeho Lee

August, 2026

1 Decoder definition

The decoder is a pure combinational function over one 16-bit input. Given a Hangul syllable code point in [U+AC00, U+D7A3], it produces the three structural axes:

$$offset = code - U+AC00, \quad i = offset/588, \quad m = (offset \bmod 588)/28, \quad f = offset \bmod 28$$

with i in $[0, 18]$, m in $[0, 20]$, and f in $[0, 27]$. There are no registers, so the latency is exactly one cycle by construction. Inputs below U+AC00 underflow and are outside the valid domain; inputs in U+D7A4..U+D7AF are structurally invalid filler positions.

2 Division structure

The constant divisions dominate the critical path. Two implementations of the same arithmetic were measured:

Structure	Implementation	Depth
Shift-subtract (naive)	Yosys techmap long division	72 logic levels
Multiply-shift (current)	$(q4 * 9363) \gg 16$ for $/28$, $(j * 781) \gg 14$ for $/21$	33 logic levels

The multiply-shift constants are exact over the valid domain: division by 28 becomes division by 7 on $q4 = offset \gg 2$ ($28 = 4 \times 7$, $q4$ at most 2792), and division by 21 applies to j at most 398. Both structures are functionally identical, which the verification pipeline proves rather than assumes.

3 Compose and distance

The decoder maps an index to the three axes. The primitive needs the inverse and a metric as well, and both are small combinational units in the same tree:

Unit	Operation	Structure
tagma_compose	axes to index, $588i + 28m + f$	two constant multiplies ($21i$ and $28p$), one validity bound per axis
tagma_dist	field-wise absolute difference of two code points	two tagma_decoder cores and three subtractors

`tagma_compose` is the decoder read backwards: the same 588 and 28 constants appear as multipliers instead of divisors, and an out-of-range axis triple is reported as `valid = 0` with `index = 0`. `tagma_dist` decodes each operand with the same core as the standalone decoder, so the two share one decode definition, and returns the three per-axis distances separately, matching `Coord::hamming_distance` on every valid pair.

4 Verification pipeline

The pipeline has a single source of truth: `tagma_core` in Rust, the reference implementation of the primitive. Every hardware artifact is checked against it. The pipeline is shown in Figure Figure 1.

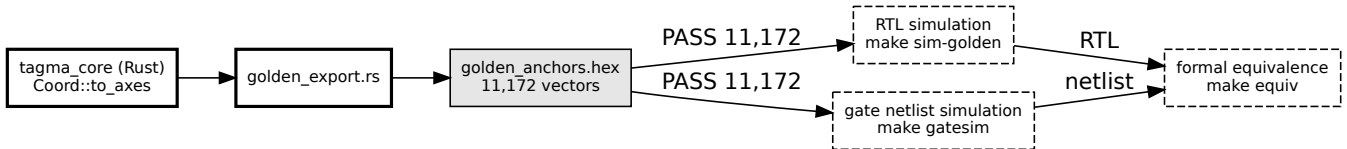


Figure 1: Hardware verification pipeline. The Rust reference is the single source of truth; every simulation and the formal equivalence check re-validate against it.

The components:

Component	Role
<code>golden_export.rs</code>	Exports all 11,172 (offset, i, m, f) vectors from <code>Coord::to_axes</code> into <code>golden_anchors.hex</code> , one packed 29-bit value per line
Formula simulation	Recomputes the formula in the testbench and compares the RTL against it
Golden simulation	Reads the anchor file and checks the RTL against the Rust reference
Gate-level simulation	Runs the synthesized netlist against the same anchor file
Formal equivalence	Proves the RTL and the netlist identical over all 2^{16} inputs with <code>equiv_simple</code> and <code>equiv_induct</code>
Compose and distance channels	Formula and golden simulation for <code>tagma_compose</code> and <code>tagma_dist</code> , plus the same gate-level and equivalence checks
Consistency gate	Python check that validates the generated anchor file against the decomposition contract

5 Results

Check	Result
Exhaustive RTL simulation (formula mode)	PASS: all 11,172 code points
Golden anchors (Rust reference mode)	PASS: all 11,172
Gate-level netlist simulation	PASS: all 11,172
Formal equivalence RTL vs netlist	proven, 50 cells, all 2^{16} inputs

Check	Result
Compose (formula mode)	PASS: all 32,768 axis combinations
Compose (golden mode)	PASS: all 11,172
Distance (formula mode)	PASS: 3 x 11,172 over varying operands and both orders
Distance (golden mode)	PASS: 3 x 11,172 over varying operands and both orders
Compose RTL vs netlist	proven, all 2^{15} axis combinations
Distance RTL vs netlist	proven, all 2^{32} input pairs

The exhaustive tests also found the boundary correction: the last valid syllable is U+D7A3, not U+D7AF. The testbench boundary was wrong initially and the failure surfaced only because the full space was enumerated.