RANGE CHECK AND RELU

1. INTRODUCTION

Range checks are a foundational component of arithmetic circuit design. They appear not only in explicit operations such as computing the maximum, minimum, or ReLU of a set of integers, but also implicitly in virtually any context where wraparound errors or overflow must be avoided. For example, to verify that $x = \max\{a,b\}$, one must ensure that $(x-a)(x-b) = 0$ and that both $x-a$ and $x-b$ are nonnegative—each of which amounts to a range check.

This document presents a general and flexible method for performing range checks in arithmetic circuits over a finite field $\mathbb{Z}/p\mathbb{Z}$, where p is a large prime. The method assumes that the integer a lies in a balanced interval of length p—for instance, $\{-(p-1)/2,\dots,(p-1)/2\}$—and verifies that a lies in a more restricted subinterval of the form

$$B - b^{\kappa} + 1 \leqslant a \leqslant B,$$

or equivalently, in an interval of the form

$$-B' \leqslant a \leqslant b^{\kappa} - 1 - B', \quad \text{where} \quad B' = b^{\kappa} - 1 - B.$$

Subject to mild and checkable conditions on B, B', b, κ, and p, the circuit can enforce such constraints reliably and efficiently.

At a high level, the idea is to reduce the range check to verifying that the shifted quantity $B-a$ (or $B'+a$) admits a base-b expansion with κ digits. This expansion is verified inside the circuit by constraining each digit d to lie in $\{0,\dots,b-1\}$. There are two options for doing this: one can enforce the constraint

$$d(d-1)\cdots(d-(b-1)) \equiv 0 \bmod p,$$

which requires a degree-b polynomial and is thus expensive when b is large, or one can check membership in a lookup table containing the valid digits. Both approaches are supported by our method, and they can be freely combined.

This flexibility allows the designer to interpolate between standard binary or base-b decomposition (small b, large κ) and a full lookup-table strategy (large b, small κ). The ability to trade off constraint cost against table size makes this technique adaptable to a wide variety of use cases and backend proving systems.

As a byproduct, we obtain a method for computing and verifying the value of a rectified linear unit (ReLU):

$$\mathrm{ReLU}(a) := \max\{0,a\}.$$

See Section 3 for details.

2. RANGE CHECK

2.1. Steps in the process. Each step in the process has a corresponding explanatory note in Subsection 2.2 that provides additional context and details.

(1) The circuit operates over the finite field $\mathbb{Z}/p\mathbb{Z}$, where p is a prime.

(2) Assume $a \in \{h-p,\dots,h-1\}$ for some integer h.

(3) To perform an upper bound check on a, use Substep (3a). For a lower bound check, use Substep (3b).

(3a) Let h and integers $b_1 \geqslant 2$, $\kappa_1 \geqslant 1$, B_1 satisfy

$$b_1^{\kappa_1} \leqslant b_1^{\kappa_1} - 1 - B_1 + h \leqslant p \quad \text{and} \quad B_1 \leqslant (b_1 - 1) b_1^{\kappa_1 - 1} \tag{2.1}$$

(3b) Let h and integers $b_2 \geqslant 2$, $\kappa_2 \geqslant 1$, B_2 satisfy

$$b_2^{\kappa_2} \leqslant B_2 + h \leqslant p \quad \text{and} \quad B_2 \leqslant (b_2 - 1) b_2^{\kappa_2 - 1} \tag{2.2}$$

(4) The goal of the circuit is to verify that a lies within a prescribed range, by enforcing one or both of the following bounds:

(4a) Upper bound:

$$a \leqslant B_1. \tag{2.3}$$

(4b) Lower bound:

$$a \geqslant -B_2. \tag{2.4}$$

(5) As most frameworks work exclusively with least residue representations, let $\bar{z}$ denote the least nonnegative residue of z mod p. The circuit receives $\bar{a}$ as an input; $\bar{B}_1$ and $\bar{B}_2$ are circuit constants, as are b_1, κ_1, b_2, κ_2.

(6) To reduce to a nonnegative range check, the circuit computes a shifted value of a:

(6a) To verify an upper bound, compute $\overline{B_1 - a} = \overline{\bar{B}_1 - \bar{a}}$.

(6b) To verify a lower bound, compute $\overline{B_2 + a} = \overline{\bar{B}_2 + \bar{a}}$.

(7) To bound a from above, see (7a). To bound a from below, see (7b).

(7a) The prover computes the κ_1 least significant base-b_1 digits $d_{1,0}, \ldots, d_{1,\kappa_1 - 1}$ of $\overline{B_1 - a}$ [Proposition 4.4, Algorithm 5.1, Listing 2], and supplies them to the circuit as a witness:

$$\overline{B_1 - a} = d_{1,0} + d_{1,1} b_1 + \cdots + d_{1,\kappa_1 - 1} b_1^{\kappa_1 - 1} + q_{1,\kappa_1} b_1^{\kappa_1}, \tag{2.5}$$

where

$$q_{1,\kappa_1} \in \mathbb{Z} \quad \text{and} \quad d_{1,0}, \ldots, d_{1,\kappa_1 - 1} \in \{0, \ldots, b_1 - 1\}, \quad 0 \leqslant i \leqslant \kappa_1 - 1. \tag{2.6}$$

(7b) The prover computes the κ_2 least significant base-b_2 digits $d_{2,0}, \ldots, d_{2,\kappa_2 - 1}$ of $\overline{B_2 + a}$ [Proposition 4.4, Algorithm 5.1, Listing 2], and supplies them to the circuit as a witness:

$$\overline{B_2 + a} = d_{2,0} + d_{2,1} b_2 + \cdots + d_{2,\kappa_2 - 1} b_2^{\kappa_2 - 1} + q_{2,\kappa_2} b_2^{\kappa_2}, \tag{2.7}$$

where

$$q_{2,\kappa_2} \in \mathbb{Z} \quad \text{and} \quad d_{2,0}, \ldots, d_{2,\kappa_1 - 1} \in \{0, \ldots, b_2 - 1\}, \quad 0 \leqslant i \leqslant \kappa_2 - 1. \tag{2.8}$$

(8) To bound a from above, see (8a). To bound a from below, see (8b).

(8a) The arithmetic circuit imposes the following constraints [Algorithm 5.3, Listing 4]:

- For $0 \leqslant i \leqslant \kappa_1 - 1$, ensure $d_{1,i} \equiv c_{1,i}$ mod p with $c_{1,i} \in \{0, \ldots, b_1 - 1\}$ *either* by requiring

$$d_{1,i}(d_{1,i} - 1) \cdots (d_{1,i} - (b_1 - 1)) \equiv 0 \text{ mod } p, \tag{2.9}$$

or by enforcing membership in a lookup table (LUT) containing the valid digits $\{0, \ldots, b_1 - 1\}$.

- Require [Algorithm 5.3, Listing 4]

$$\overline{B_1 - a} \equiv d_{1,0} + d_{1,1} b_1 + \cdots + d_{1,\kappa_1 - 1} b_1^{\kappa_1 - 1} \text{ mod } p. \tag{2.10}$$

(8b) The arithmetic circuit imposes the following constraints [Algorithm 5.3, Listing 4]:

- For $0 \leqslant i \leqslant \kappa_2 - 1$, ensure $d_{2,i} \equiv c_{2,i}$ mod p with $c_{2,i} \in \{0, \ldots, b_2 - 1\}$ *either* by requiring

$$d_{2,i}(d_{2,i} - 1) \cdots (d_{2,i} - (b_2 - 1)) \equiv 0 \text{ mod } p, \tag{2.11}$$

or by enforcing membership in a lookup table (LUT) containing the valid digits $\{0, \ldots, b_2 - 1\}$.

- Require [Algorithm 5.3, Listing 4]

$$\overline{B_2 + a} \equiv d_{2,0} + d_{2,1}b_2 + \cdots + d_{2,\kappa_2-1}b_2^{\kappa_2-1} \bmod p. \tag{2.12}$$

(9) To bound a from above, see (9a). To bound a from below, see (9b).

(9a) Assuming $a \in \{h-p, \ldots, h-1\}$ and that (2.1) holds, these constraints guarantee [Corollary 4.2] that

$$B_1 - a = d_{1,0} + d_{1,1}b_1 + \cdots + d_{1,\kappa_1-1}b_1^{\kappa_1-1}, \tag{2.13}$$

provided each $d_{1,i}$ is interpreted as a least nonnegative residue modulo p, and hence that

$$B_1 - b_1^{\kappa_1} + 1 \leqslant a \leqslant B_1. \tag{2.14}$$

Conversely, if (2.14) holds, then the constraints of Step (8a) also hold (i.e., there exist digits $d_{1,i} \in \{0, \ldots, b_1 - 1\}$ such that (2.10) holds).

(9b) Assuming $a \in \{h-p, \ldots, h-1\}$ and that (2.2) holds, these constraints guarantee [Corollary 4.3] that

$$B_2 + a = d_{2,0} + d_{2,1}b_2 + \cdots + d_{2,\kappa_2-1}b_2^{\kappa_2-1}, \tag{2.15}$$

provided each $d_{2,i}$ is interpreted as a least nonnegative residue modulo p, and hence that

$$-B_2 \leqslant a \leqslant b_2^{\kappa_2} - 1 - B_2. \tag{2.16}$$

Conversely, if (2.16) holds, then the constraints of Step (8b) also hold (i.e., there exist digits $d_{2,i} \in \{0, \ldots, b_2 - 1\}$ such that (2.12) holds).

2.2. Commentary on each step

(1) Typically, p is an n-bit prime satisfying

$$2^{n-1} \leqslant p < 2^n,$$

with $n \approx 256$. In practice, we use the prime field associated with the scalar field of the BN254 elliptic curve, a 254-bit prime that offers a good balance between security and efficiency. This field is widely supported in cryptographic applications and zk-SNARK frameworks due to the curve's pairing-friendly properties and efficient arithmetic over $\mathbb{Z}/p\mathbb{Z}$.

Polyhedra Network's EXPANDER prover and EXPANDERCOMPILERCOLLECTION (ECC) [1, 2, 3] support both the BN254 prime and the 31-bit Mersenne prime $2^{31} - 1$. Although the latter is far too small for cryptographic security on its own, ECC uses internal field extensions to enable practical use of this field in production settings, trading off cryptographic hardness for performance and recursion flexibility in certain use cases.

(2) A natural and common choice is $h = (p+1)/2$, in which case $\{h-p, \ldots, h-1\}$ simplifies to the balanced interval $\{-(p-1)/2, \ldots, (p-1)/2\}$. This choice is appropriate when off-circuit reasoning does not support a more specific assumption about the sign or magnitude of a.

The assumption that $a \in \{h-p, \ldots, h-1\}$ is essential and must be justified externally, since the circuit itself can only constrain the least residue representative $\bar{a}$ of a mod p. To draw conclusions about the original integer a, it is necessary to exclude all other integers congruent to $\bar{a}$ modulo p. This assumption is also an indispensable hypothesis of Corollaries 4.3 and 4.2, which are invoked in Step (9).

If $0 \leqslant h \leqslant p$, then the relationship between a and its least residue $\bar{a}$ is fully determined by

$$\bar{a} = \begin{cases} a & \text{if } a \in \{0, \ldots, h-1\}, \\ a + p & \text{if } a \in \{h-p, \ldots, -1\}, \end{cases} \quad \text{and} \quad a = \begin{cases} \bar{a} & \text{if } \bar{a} \in \{0, \ldots, h-1\}, \\ \bar{a} - p & \text{if } \bar{a} \in \{h, \ldots, p-1\}. \end{cases} \tag{2.17}$$

This correspondence allows us to reason unambiguously about a based on $\bar{a}$ within the circuit.

(3) The parameters b_1, κ_1, B_1 (resp. b_2, κ_2, B_2), and h are fixed at circuit compilation time and are therefore treated as constants of the circuit. However, only the least nonnegative residues $\bar{B}_1$ and $\bar{B}_2$ (modulo p) are used internally by the circuit, since it operates over the finite field $\mathbb{Z}/p\mathbb{Z}$.

Note that B_1 and B_2 themselves need not be nonnegative. The inequalities (2.1) and (2.2) imply only that

$$h - p + b_1^{\kappa_1} - 1 \leqslant B_1 \leqslant (b_1 - 1)b_1^{\kappa_1-1} \quad \text{and} \quad b_2^{\kappa_2} - h \leqslant B_2 \leqslant (b_2 - 1)b_2^{\kappa_2-1}.$$

Since arithmetic is carried out modulo p, the circuit treats $\bar{B}_1$ and $\bar{B}_2$ as constants and uses them in computations involving $\bar{a}$.

When lookup tables are not employed in Step (8), it is common to set $b_1 = 2$ (resp. $b_2 = 2$) to minimize the cost of enforcing digit validity. Nonetheless, there is no inherent restriction preventing one from selecting $\kappa_1 = 1$ (resp. $\kappa_2 = 1$) and a correspondingly large value of b_1 (resp. b_2).

(4) The actual verification logic in the circuit does not enforce the upper bound (2.3) directly. Instead, it ensures that

$$B_1 - b_1^{\kappa_1} + 1 \leqslant a \leqslant B_1,$$

based on the existence of a valid base-b_1 decomposition of $B_1 - a$. This means that if $a \leqslant B_1 - b_1^{\kappa_1}$, the check will fail, despite a satisfying the desired inequality $a \leqslant B_1$.

To ensure the range check does not inadvertently exclude valid values, the parameters should be chosen so that all expected values of a fall within the interval $[B_1 - b_1^{\kappa_1} + 1, B_1]$. From (2.1), it follows that

$$B_1 - b_1^{\kappa_1} + 1 \leqslant 1 - b_1^{\kappa_1 - 1},$$

so it suffices to choose b_1, κ_1, and B_1 such that all valid a lie in the interval $[1 - b_1^{\kappa_1-1}, B_1]$.

An analogous situation arises for the lower bound. The circuit does not enforce $a \geqslant -B_2$ directly, but instead verifies

$$-B_2 \leqslant a \leqslant b_2^{\kappa_2} - 1 - B_2,$$

based on a base-b_2 decomposition of $B_2 + a$. Assumption (2.2) guarantees that

$$b_2^{\kappa_2} - 1 - B_2 \geqslant b_2^{\kappa_2 - 1} - 1,$$

so it suffices to choose b_2, κ_2, and B_2 such that all valid a lie in the interval $[-B_2, b_2^{\kappa_2-1} - 1]$.

(5) This step formalizes the notational convention introduced in Comment (2): all values z handled by the circuit are represented by their least nonnegative residues modulo p, denoted $\bar{z}$. In particular, the input $\bar{a}$ and constants $\bar{B}_1$, $\bar{B}_2$ are understood to satisfy

$$\bar{a} \equiv a \bmod p, \qquad \bar{B}_1 \equiv B_1 \bmod p, \qquad \bar{B}_2 \equiv B_2 \bmod p,$$

with $a, B_1, B_2 \in \{h-p, \ldots, h-1\}$ ensuring a unique correspondence. This convention allows all subsequent computations to occur within the field $\mathbb{Z}/p\mathbb{Z}$, consistent with the assumptions of most SNARK-compatible arithmetic circuit frameworks.

(6) The circuit receives $\bar{a}$ as input; $\bar{B}_1$ and $\bar{B}_2$ are circuit constants. The shifted values

$$\bar{B}_1 - \bar{a} \quad \text{and} \quad \bar{B}_2 + \bar{a}$$

are computed directly as field operations in $\mathbb{Z}/p\mathbb{Z}$. Because arithmetic is performed modulo p, the results are automatically interpreted as least nonnegative residues, regardless of the original integer representatives of a, B_1, or B_2.

(7) This step uses the standard method of base-b expansion via repeated Euclidean division, applied off-circuit by the prover. The digit decomposition of $\overline{B_1 - a}$ (resp. $\overline{B_2 + a}$) is not constrained directly by the circuit at this point; instead, the prover computes and provides the digits as part of the witness.

The extra term $q_{1,\kappa_1} b_1^{\kappa_1}$ (resp. $q_{2,\kappa_2} b_2^{\kappa_2}$) accommodates the possibility that $\overline{B_1 - a}$ (resp. $\overline{B_2 + a}$) exceeds $b_1^{\kappa_1} - 1$ (resp. $b_2^{\kappa_2} - 1$). However, when the constraints of Step (8) are satisfied—specifically, when a lies within the interval guaranteed by Corollary 4.2 (resp. 4.3)—then $\overline{B_1 - a} \in [0, b_1^{\kappa_1} - 1]$ (resp. $\overline{B_2 + a} \in [0, b_2^{\kappa_2} - 1]$), and so we have $q_{1,\kappa_1} = 0$ (resp. $q_{2,\kappa_2} = 0$).

Nevertheless, because the circuit verifies constraints only modulo p, the presence of the q-term is formally allowed at this stage and will be rendered irrelevant during the constraint checks of Step (8).

(8) Each digit constraint of the form

$$d_{1,i}(d_{1,i} - 1) \cdots (d_{1,i} - (b_1 - 1)) \equiv 0 \bmod p$$

ensures that $d_{1,i}$ is one of the b_1 valid digits in $\{0, \ldots, b_1 - 1\}$. However, this constraint is a degree-b_1 polynomial, which naïvely incurs $b_1 - 1$ multiplication gates per digit. When $b_1 \geqslant 3$, this is expensive. For this reason, it is preferable to verify digit validity via membership in a precomputed lookup table (LUT), especially when a dedicated hint mechanism or polynomial commitment is available.

The same considerations apply to the lower-bound case using $d_{2,i}$ and base b_2. Circuit designers may choose different strategies for upper and lower bounds depending on performance trade-offs, especially when only one bound is active in a given application.

(9) The congruence in (2.10) is equivalent to

$$B_1 - a \equiv d_{1,0} + d_{1,1} b_1 + \cdots + d_{1,\kappa_1 - 1} b_1^{\kappa_1 - 1} \bmod p.$$

However, we express the left-hand side using the notation $\overline{B_1 - a}$ to emphasize that all quantities within the circuit are interpreted as least residues modulo p.

It is important to clarify that the digit validity constraint (2.9) does not directly enforce that $d_{1,i} \in \{0, \dots, b_1 - 1\}$, but rather that $d_{1,i} \equiv c_{1,i} \bmod p$ for some $c_{1,i}$ in this range. Since circuit values are typically represented as canonical integers in $\{0, \dots, p-1\}$, this congruence ensures that $d_{1,i} = c_{1,i}$ in practice. Thus, the constraint effectively enforces digit validity within the finite field representation.

The converse direction in Step (9a) relies on the forward implication of Corollary 4.2, and its correctness depends on the parameter conditions ensuring a lies in the appropriate range. See also Comment (4) for additional discussion.

Analogous reasoning applies for the lower bound.

3. ReLU

In the case where $B_1 = (b_1 - 1)b_1^{\kappa_1 - 1}$ (resp. $B_2 = (b_2 - 1)b_2^{\kappa_2 - 1}$), the sign of a is completely determined by the most significant base-b digit $d_{1,\kappa_1 - 1}$ of $\overline{B_1 - a}$ (resp. $d_{2,\kappa_2 - 1}$ of $\overline{B_2 + a}$). Thus, after verifying the base-b decomposition, the circuit can compute

$$\mathrm{ReLU}(a) := \max\{0, a\},$$

as follows. First, compute

$$\mathrm{sign}(a) = \mathbf{1}_{\{d_{\kappa_1 - 1} < b_1 - 1\}} = \begin{cases} 1 & \text{if } d_{\kappa_1 - 1} < b_1 - 1, \\ 0 & \text{otherwise,} \end{cases} \tag{3.1}$$

or

$$\mathrm{sign}(a) = \mathbf{1}_{\{d_{\kappa_2 - 1} = b_2 - 1\}} = \begin{cases} 1 & \text{if } d_{\kappa_2 - 1} = b_2 - 1, \\ 0 & \text{otherwise.} \end{cases} \tag{3.2}$$

Then, by Corollary 4.2 and 4.3,

$$\mathrm{ReLU}(a) = \mathrm{sign}(a) \cdot \bar{a}. \tag{3.3}$$

We emphasize that this is subject to the assumptions of Steps (2) and (3) of the process in Subsection 2.1.

No additional constraint is needed to enforce correctness of (3.3), as it is computed within the circuit itself, based on a base-b decomposition that has already been verified. However, if we wish to prove that $\mathrm{ReLU}(a)$ matches an externally provided value or a public input, we must include an explicit equality constraint, as described below.

3.1. Steps in the process. As the process is essentially a range check with one extra step, we only provide a corresponding explanatory note for this final step, in Subsection 3.2.

(1) The circuit operates over the finite field $\mathbb{Z}/p\mathbb{Z}$, where p is a prime.

(2) Assume $a, y \in \{h - p, \dots, h - 1\}$ for some integer h.

(3) The goal of the circuit is to verify that

$$y = \mathrm{ReLU}(a). \tag{3.4}$$

(4) As most frameworks work exclusively with least residue representations, let $\bar{z}$ denote the least nonnegative residue of $z \bmod p$. The circuit receives $\bar{y}$ and $\bar{a}$ as inputs.

(5) We can perform Steps (5a) and (6a), or Steps (5b) and (6b).

(5a) Follow Steps (3a) through (9a) for upper-bounding a by B_1, with

$$B_1 = (b_1 - 1)b_1^{\kappa_1 - 1}. \tag{3.5}$$

Note that this choice, together with the first pair of inequalities in (2.1) imply that

$$1 + (b_1 - 1)b_1^{\kappa_1 - 1} \leqslant h \leqslant p + 1 - b_1^{\kappa_1}. \tag{3.6}$$

(5b) Follow Steps (3b) through (9b) for lower-bounding a by $-B_2$, with

$$B_2 = (b_2 - 1)b_2^{\kappa_2 - 1}. \tag{3.7}$$

Note that this choice, together with the first pair of inequalities in (2.2) imply that

$$b_2^{\kappa_2 - 1} \leqslant h \leqslant p - (b_2 - 1)b_2^{\kappa_2 - 1}. \tag{3.8}$$

(6) (6a) By Corollary 4.2,

$$\mathrm{ReLU}(a) = \mathbf{1}_{\{d_{\kappa_1-1} < b_1 - 1\}} \bar{a}. \tag{3.9}$$

The right-hand side is computed within the circuit, based on a verified base-b decomposition. If $\bar{y}$ is provided externally or as a public input, the circuit must impose the following constraint:

$$\bar{y} \equiv \mathbf{1}_{\{d_{\kappa_1-1} < b_1 - 1\}} \bar{a} \bmod p. \tag{3.10}$$

(6b) By Corollary 4.3,

$$\mathrm{ReLU}(a) = \mathbf{1}_{\{d_{\kappa_2-1} = b_2 - 1\}} \bar{a}. \tag{3.11}$$

The right-hand side is computed within the circuit, which then imposes the constraint

$$\bar{y} \equiv \mathbf{1}_{\{d_{\kappa_2-1} = b_2 - 1\}} \bar{a} \bmod p. \tag{3.12}$$

(7) Since $a, y \in \{h-p, \dots, h-1\}$ and $1 \leqslant h \leqslant p$, this ensures that (3.4) holds.

3.2. Commentary on last step

(7) Note that (3.6) and (3.8) both imply that $1 \leqslant h \leqslant p$, which, since $y \in \{h-p, \dots, h-1\}$, in turn implies that

$$-(p-1) \leqslant y \leqslant p-1. \tag{3.13}$$

Recall that $a \in \{h-p, \dots, h-1\}$ as well. Now, if $a < 0$, then $\mathrm{ReLU}(a) = 0$, so (3.9) and (3.10) (resp. (3.11) and (3.12)) imply that $y \equiv 0 \bmod p$. In view of (3.13), we must conclude that $y = 0$. Hence (3.4) holds.

On the other hand, if $a \geqslant 0$, then $\mathrm{ReLU}(a) = a$, so (3.9) and (3.10) (resp. (3.11) and (3.12)) imply that $y - a \equiv 0 \bmod p$. Since $y \in \{h-p, \dots, h-1\}$ and $a \in \{0, \dots, h-1\}$, we have

$$h - p - (h-1) \leqslant y - a \leqslant h - 1 - (h-1) \quad \Rightarrow \quad -(p-1) \leqslant y - a \leqslant 0,$$

leaving $y - a = 0$ as the only possibility. Hence $y = a$ and (3.4) holds.

4. The maths

Proposition 4.1. *Let $p \geqslant 2$ be a modulus, and let $a \in \{h-p, \dots, h-1\}$ for some integer h. Let integers U, V satisfy*

$$U + V \leqslant U + h \leqslant p.$$

The following holds for both $T = U + a$ and $T = V - 1 - a$. Write $\overline{T}$ for the least nonnegative residue of T mod p. If

$$\overline{T} \leqslant U + V - 1,$$

then $\overline{T} = T$, and hence

$$-U \leqslant a \leqslant V - 1.$$

Proof. Since $\overline{T} \equiv T \bmod p$, there exists an integer t such that $tp = T - \bar{T}$. From $0 \leqslant \bar{T} \leqslant U + V - 1$, it follows that

$$T - (U + V - 1) \leqslant tp \leqslant T.$$

From $a \leqslant h - 1$ and $U + h \leqslant p$, we obtain

$$U + a \leqslant U + h - 1 \leqslant p - 1.$$

Therefore,

$$V - 1 - a - (U + V - 1) = -(U + a) \geqslant -(p-1).$$

From $a \geqslant h - p$ and $U + V \leqslant U + h$, we obtain

$$V - 1 - a \leqslant h - 1 - a \leqslant h - 1 - (h - p) = p - 1.$$

Therefore,

$$U + a - (U + V - 1) = -(V - 1 - a) \geqslant -(p-1).$$

Whether $T = U + a$ or $T = V - 1 - a$, we conclude

$$-(p-1) \leqslant T - (U + V - 1) \leqslant tp \leqslant T \leqslant p - 1,$$

and hence

$$-\frac{p-1}{p} \leqslant t \leqslant \frac{p-1}{p}.$$

As t is an integer, it follows that $t = 0$, and therefore $T = \overline{T}$. The inequality $\overline{T} \leqslant U + V - 1$ now gives

$$T \leqslant U + V - 1 \quad \Rightarrow \quad \begin{cases} a \leqslant V - 1 & \text{if } T = U + a, \\ -a \leqslant U & \text{if } T = V - 1 - a. \end{cases}$$

Since $\overline{T} \geqslant 0$, we also have $U \geqslant -a$ in the case $T = U + a$, and $V - 1 \geqslant a$ in the case $T = V - 1 - a$. In either case, we obtain $-U \leqslant a \leqslant V - 1$. □

Corollary 4.2. *Let $p \geqslant 2$ be a modulus, and let $a \in \{h-p, \dots, h-1\}$ for some integer h. Suppose h and integers $b \geqslant 2$, $\kappa \geqslant 1$, R satisfy*

$$b^\kappa \leqslant b^\kappa - 1 - R + h \leqslant p \quad \textit{and} \quad R \leqslant (b-1)b^{\kappa-1}.$$

Then

$$R - b^\kappa + 1 \leqslant a \leqslant R$$

if and only if there exist digits $d_0, \dots, d_{\kappa-1} \in \{0, \dots, b-1\}$ such that

$$R - a \equiv d_0 + d_1 b + \cdots + d_{\kappa-1} b^{\kappa-1} \bmod p.$$

Moreover, if $R = (b-1)b^{\kappa-1}$, then $a \leqslant 0$ if and only if $d_{\kappa-1} = b - 1$.

Proof. Suppose $R - b^\kappa + 1 \leqslant a \leqslant R$. Then

$$0 \leqslant R - a \leqslant b^\kappa - 1,$$

so $R - a$ has a unique base-b expansion

$$d_0 + d_1 b + \cdots + d_{\kappa-1} b^{\kappa-1}$$

with digits $d_0, \dots, d_{\kappa-1} \in \{0, \dots, b-1\}$. In particular,

$$R - a \equiv d_0 + d_1 b + \cdots + d_{\kappa-1} b^{\kappa-1} \bmod p.$$

Conversely, suppose this congruence holds for some such digits. Since the right-hand side lies in $[0, b^\kappa - 1] \subseteq [0, p-1]$, it must be the least nonnegative residue of $R - a$ mod p. By Proposition 4.1, applied with $U = b^\kappa - 1 - R$, $V - 1 = R$, and $T = V - 1 - a$, we conclude that

$$R - a = d_0 + d_1 b + \cdots + d_{\kappa-1} b^{\kappa-1},$$

and hence $0 \leqslant R - a \leqslant b^\kappa - 1$, i.e. $R - b^\kappa + 1 \leqslant a \leqslant R$.

Now suppose $R = (b-1)b^{\kappa-1}$. Then

$$R - a = d_0 + d_1 b + \cdots + d_{\kappa-1} b^{\kappa-1} \quad \Rightarrow \quad a = [b - 1 - d_{\kappa-1}]b^{\kappa-1} - (d_0 + d_1 b + \cdots + d_{\kappa-2} b^{\kappa-2}).$$

If $d_{\kappa-1} = b - 1$, then $a \leqslant 0$. Otherwise, $b - 1 - d_{\kappa-1} - (b-1) \geqslant 1$, so

$$a \geqslant b^{\kappa-1} - (b-1)(1 + b + \cdots + b^{\kappa-2}) = 1.$$

□

Corollary 4.3. *Let $p \geqslant 2$ be a modulus, and let $a \in \{h-p, \dots, h-1\}$ for some integer h. Suppose h and integers $b \geqslant 2$, $\kappa \geqslant 1$, S satisfy*

$$b^\kappa \leqslant S + h \leqslant p \quad \textit{and} \quad S \leqslant (b-1)b^{\kappa-1}.$$

Then

$$-S \leqslant a \leqslant b^\kappa - 1 - S$$

if and only if there exist digits $d_0, \dots, d_{\kappa-1} \in \{0, \dots, b-1\}$ such that

$$S + a \equiv d_0 + d_1 b + \cdots + d_{\kappa-1} b^{\kappa-1} \bmod p.$$

Moreover, if $S = (b-1)b^{\kappa-1}$, then $a \geqslant 0$ if and only if $d_{\kappa-1} = b - 1$.

Proof. Suppose $-S \leqslant a \leqslant b^\kappa - 1 - S$. Then

$$0 \leqslant S + a \leqslant b^\kappa - 1,$$

so $S + a$ has a unique base-b expansion

$$d_0 + d_1 b + \cdots + d_{\kappa-1} b^{\kappa-1}$$

with digits $d_0, \ldots, d_{\kappa-1} \in \{0, \ldots, b-1\}$. In particular,

$$S + a \equiv d_0 + d_1 b + \cdots + d_{\kappa-1} b^{\kappa-1} \bmod p.$$

Conversely, suppose this congruence holds for some such digits. Since the right-hand side lies in $[0, b^\kappa - 1] \subseteq [0, p-1]$, it must be the least nonnegative residue of $S + a$ mod p. By Proposition 4.1, applied with $U = S$, $V = b^\kappa - S$, and $T = U + a$, we conclude that

$$S + a = d_0 + d_1 b + \cdots + d_{\kappa-1} b^{\kappa-1},$$

and hence $0 \leqslant S + a \leqslant b^\kappa - 1$, i.e. $-S \leqslant a \leqslant b^\kappa - 1 - S$.

Now suppose $S = (b-1)b^{\kappa-1}$. Then

$$S + a = d_0 + d_1 b + \cdots + d_{\kappa-1} b^{\kappa-1} \quad \Rightarrow \quad a = d_0 + d_1 b + \cdots + d_{\kappa-2} b^{\kappa-2} + [d_{\kappa-1} - (b-1)] b^{\kappa-1}.$$

If $d_{\kappa-1} = b - 1$, then $a \geqslant 0$. Otherwise, $d_{\kappa-1} - (b-1) \leqslant -1$, so

$$a \leqslant (b-1)(1 + b + \cdots + b^{\kappa-2}) - b^{\kappa-1} = -1.$$

□

Proposition 4.4. *Fix an integer $b \geqslant 2$. Let q_0 be any integer. For $0 \leqslant i \leqslant \kappa - 1$, let q_{i+1} and d_i be the unique integers satisfying $q_i = b q_{i+1} + d_i$ and $d_i \in \{0, 1, \ldots, b-1\}$.*

(a) Then

$$q_0 = d_0 + d_1 b + \cdots + d_{\kappa-1} b^{\kappa-1} + q_\kappa b^\kappa. \tag{4.1}$$

(b) The following statements are equivalent: (i) $0 \leqslant q_0 \leqslant b^\kappa - 1$; (ii) $q_\kappa = 0$; (iii) $(d_0, \ldots, d_{\kappa-1})$ is the κ-digit base-b representation of q_0 (with d_0 the least significant digit).

Proof. (a) We induct on κ. The result holds trivially for $\kappa = 0$. Suppose the result holds with $\kappa = n$ for some $n \geqslant 0$. Now consider $\kappa = n + 1$. We have $q_i = b q_{i+1} + d_i$, $d_i \in \{0, 1, \ldots, b-1\}$ for $0 \leqslant i \leqslant n - 1$ and also $i = n$. By inductive hypothesis,

$$\begin{aligned} q_0 &= d_0 + d_1 b + \cdots + d_{n-1} b^{n-1} + q_n b^n \\ &= d_0 + d_1 b + \cdots + d_{n-1} b^{n-1} + (b q_{n+1} + d_n) b^n \\ &= d_0 + d_1 b + \cdots + d_n b^n + q_{n+1} b^{n+1}. \end{aligned}$$

(b) Suppose $0 \leqslant q_0 \leqslant b^\kappa - 1$. In view of (4.1), this implies

$$q_\kappa b^\kappa = q_0 - \left(d_0 + d_1 b + \cdots + d_{\kappa-1} b^{\kappa-1} + q_\kappa b^\kappa\right) \leqslant q_0 \leqslant b^\kappa - 1.$$

Hence $q_\kappa \leqslant 1 - b^{-\kappa} < 1$. Also,

$$-q_\kappa b^\kappa = \left(d_0 + d_1 b + \cdots + d_{\kappa-1} b^{\kappa-1} + q_\kappa b^\kappa\right) - q_0 \leqslant (b-1)\left(1 + b + \cdots + b^{\kappa-1}\right) = b^\kappa - 1.$$

Hence $q_\kappa \geqslant b^{-\kappa} - 1 > -1$. Since q_κ is an integer, we must therefore have $q_\kappa = 0$, which implies

$$q_0 = d_0 + d_1 b + \cdots + d_{\kappa-1} b^{\kappa-1} + q_\kappa b^\kappa,$$

i.e., $(d_0, \ldots, d_{\kappa-1})$ is the κ-digit base-b representation of q_0. Finally, if this holds, then $0 \leqslant q_0 \leqslant b^\kappa - 1$, because the right-hand side lies between 0 and $(b-1)(1 + b + \cdots + b^{\kappa-1}) = b^\kappa - 1$. □

5. The code

Rust code in this section is designed for implementation within the ExpanderCompilerCollection (ECC) framework [3, 4]. This library provides a specialized interface for constructing and verifying arithmetic circuits.

5.1. The κ least significant base-b digits of a nonnegative integer. Whether performing a range check for nonnegative integers, a signed range check, or verifying a ReLU computation, it is essential to compute the κ least significant base-b digits of a nonnegative integer. We provide pseudocode for this task, with correctness justified by Proposition 4.4, followed by Rust implementations for both the special case $b = 2$ and the general case $b \geqslant 2$.

Algorithm 5.1 `to_base_b`: compute the κ least significant digits of base-b representation of a nonnegative integer

```
Require: nonnegative integer $q_0$, integer base $b \geqslant 2$, and integer $\kappa \geqslant 1$
Ensure: list $d = [d_0, \ldots, d_{\kappa-1}]$ such that $d_i$ is the $i$-th least significant base-$b$ digit of $q_0$
1: $d \leftarrow []$                          ▷ Initialize empty list for digits
2: $q \leftarrow q_0$
3: for $i \leftarrow 0$ to $\kappa - 1$ do
4:     append $q$ mod $b$ to $d$
5:     $q \leftarrow \lfloor q/b \rfloor$                 ▷ Drop the extracted digit
6: end for
7: return $d$
```

```rust
fn to_binary<C: Config>(api: &mut API<C>, q_0: Variable, kappa: usize) -> Vec<Variable> {
    let mut d = Vec::with_capacity(kappa); // Preallocate vector
    let mut q = q_0; // Copy q_0 to modify iteratively

    for _ in 0..kappa {
        d.push(api.unconstrained_bit_and(q, 1)); // Extract least significant bit
        q = api.unconstrained_shift_r(q, 1); // Shift right by 1 bit
    }

    d
}
```

Listing 1: ECC Rust API: compute the κ least significant bits of binary representation of a nonnegative integer

```rust
/// Computes the `kappa` least significant digits of the base-`b` representation of `q_0`.
/// Assumes `q_0` is a nonnegative integer and `b >= 2`.
fn to_base_b<C: Config, Builder: RootAPI<C>>(
    api: &mut Builder,
    q_0: Variable,
    b: u32,
    kappa: usize,
) -> Vec<Variable> {
    assert!(b >= 2, "Base b must be at least 2");

    let mut d = Vec::with_capacity(kappa);
    let mut q = q_0;

    for _ in 0..kappa {
        let digit = api.unconstrained_mod(q, b);
        d.push(digit);
        q = api.unconstrained_int_div(q, b);
    }

    d
}
```

Listing 2: ECC Rust API: compute the κ least significant base-b digits of a nonnegative integer

5.2. Reconstructing an integer from its base-b representation and imposing constraints to ensure correctness. We provide pseudocode for reconstructing a nonnegative integer from its base-b representation. At the same time, we impose constraints to ensure the validity of the representation, with correctness justified by Corollaries 4.2 and 4.3. In the pseudocode, each assertion represents a constraint to be enforced within the circuit.

We begin with the special case $b = 2$, where we impose the constraint $d_i(d_i - 1) \equiv 0 \bmod p$ to ensure that each d_i is a valid binary digit. For the general case $b \geqslant 2$, we use a lookup table to enforce that each digit lies in the valid set $\{0, 1, \ldots, b-1\}$. Finally, to complete the range check, we assert that the original integer is equal to its reconstructed value, thereby linking the digit representation to the actual input being verified.

Algorithm 5.2 `from_binary`: reconstruct and verify a nonnegative integer from at most κ least significant bits

```
Require: list of binary digits d, nonnegative integer $\kappa$, and original value x
Ensure: reconstructed_integer: the integer represented by the first $\kappa$ bits of d
 1: reconstructed_integer $\leftarrow 0$
 2: for $i \leftarrow 0$ to $\max\{\kappa-1, \mathrm{len}(d)-1\}$ do
 3:     bit $\leftarrow d[i]$                                    ▷ Binary digit check: ensure bit $\in \{0,1\}$
 4:     bit_minus_one $\leftarrow$ bit $- 1$
 5:     bit_by_bit_minus_one $\leftarrow$ bit $\times$ bit_minus_one
 6:     assert bit_by_bit_minus_one $= 0$
 7:     bit_by_two_to_the_i $\leftarrow$ bit $\times 2^i$
 8:     reconstructed_integer $\leftarrow$ reconstructed_integer $+$ bit_by_two_to_the_i
 9: end for
10: assert reconstructed_integer $=$ x           ▷ Final constraint: confirm correctness of reconstruction
11: return reconstructed_integer
```

Algorithm 5.3 `from_base_b`: reconstruct and verify a nonnegative integer from at most κ least significant base-b digits

```
Require: list of base-$b$ digits d, nonnegative integer $\kappa$, and original value x
Ensure: reconstructed_integer: the integer represented by the first $\kappa$ base-$b$ digits of d
 1: reconstructed_integer $\leftarrow 0$
 2: LOOKUP_TABLE $\leftarrow \{0, 1, \ldots, b-1\}$                       ▷ Predefined valid digit set
 3: for $i \leftarrow 0$ to $\max\{\kappa-1, \mathrm{len}(d)-1\}$ do
 4:     digit $\leftarrow d[i]$
 5:     Enforce digit $\in$ LOOKUP_TABLE as a circuit constraint
 6:     digit_by_b_to_the_i $\leftarrow$ digit $\times b^i$
 7:     reconstructed_integer $\leftarrow$ reconstructed_integer $+$ digit_by_b_to_the_i
 8: end for
 9: assert reconstructed_integer $=$ x    ▷ Final constraint: ensure correctness of base-$b$ representation
10: return reconstructed_integer
```

```rust
fn binary_digit_check<C: Config>(api: &mut API<C>, d: &[Variable]) {
    for &bit in d.iter() {
        let bit_minus_one = api.sub(1, bit);
        let bit_by_bit_minus_one = api.mul(bit, bit_minus_one);
        api.assert_is_zero(bit_by_bit_minus_one);
    }
}

fn from_binary<C: Config>(api: &mut API<C>, d: &[Variable], kappa: usize, x: Variable) -> Variable
 {
    binary_digit_check(api, d);
    let mut reconstructed_integer = api.constant(0);

    for (i, &bit) in d.iter().take(kappa).enumerate() {
        let bit_by_two_to_the_i = api.mul(1 << i, bit);
        reconstructed_integer = api.add(reconstructed_integer, bit_by_two_to_the_i);
    }

    api.assert_is_equal(reconstructed_integer, x);

    reconstructed_integer
}
```

Listing 3: ECC Rust API: reconstruct a nonnegative integer from at most κ least significant bits and impose constraints

The code below integrates ECC's LogUp circuit; see [5] for documentation. Although untested and subject to revision, it provides a working draft to build upon through further experimentation and refinement.

```rust
fn lookup_digit<C: Config, API: RootAPI<C>>(
    api: &mut API,
    digit: Variable,
    lookup_table: &mut LogUpSingleKeyTable
) {
    // Use the lookup table (populated with valid digit constants) to constrain ‘digit‘.
    // The second argument here is the associated value vector, which in this case we assume to be
     empty.
    lookup_table.query(digit, vec![]);
}

// Check that every digit in the slice 'd' is a valid base-b digit using the lookup table.
fn base_b_digit_check<C: Config, API: RootAPI<C>>(
    api: &mut API,
    d: &[Variable],
    b: u32,
    lookup_table: &mut LogUpSingleKeyTable
) {
    for &digit in d.iter() {
        lookup_digit(api, digit, lookup_table);
    }
}

// Reconstruct an integer from the first 'kappa' digits in 'd' (assumed little-endian)
// and enforce that each digit is a valid base-b digit via the lookup table.
// Finally, assert equality with a given input 'x'.
fn from_base_b<C: Config, API: RootAPI<C>>(
    api: &mut API,
    d: &[Variable],
    kappa: usize,
    b: u32,
    lookup_table: &mut LogUpSingleKeyTable,
    x: Variable
) -> Variable {
    base_b_digit_check(api, d, b, lookup_table);

    let mut reconstructed_integer = api.constant(0);
    for (i, &digit) in d.iter().take(kappa).enumerate() {
        let factor = api.constant(b.pow(i as u32));
        let term = api.mul(factor, digit);
        reconstructed_integer = api.add(reconstructed_integer, term);
    }

    api.assert_is_equal(reconstructed_integer, x);

    reconstructed_integer
}
```
Listing 4: ECC Rust API: reconstruct a nonnegative integer from at most κ least significant base-b digits and impose constraints

5.3. Range check. We now complete the range check process by computing the least residue representations

$$x_1 = \overline{B_1 - a}, \qquad x_2 = \overline{B_2 + a},$$

where $\bar{a}$, $\bar{B}_1$, and $\bar{B}_2$ are least nonnegative residues modulo p, and $a \in \{h-p,\dots,h-1\}$. The parameters $b_1, b_2 \geqslant 2$, $\kappa_1, \kappa_2 \geqslant 1$, and $B_1, B_2 \in \mathbb{Z}$ must satisfy:

$$\begin{aligned} b_1^{\kappa_1} &\leqslant b_1^{\kappa_1} - 1 - B_1 + h \leqslant p, & B_1 &\leqslant (b_1 - 1)b_1^{\kappa_1 - 1}, \\ b_2^{\kappa_2} &\leqslant B_2 + h \leqslant p, & B_2 &\leqslant (b_2 - 1)b_2^{\kappa_2 - 1}. \end{aligned}$$

The values $\bar{B}_1$ and $\bar{B}_2$ are computed externally and passed to the circuit as constants. Inside the circuit, we compute $x_1 = \bar{B}_1 - \bar{a}$ and $x_2 = \bar{B}_2 + \bar{a}$, and then apply the digit decomposition and reconstruction procedure to verify that each quantity lies in the appropriate range. This step ensures that $a \in [B_1 - b_1^{\kappa_1} + 1, B_1]$ or $a \in [-B_2, b_2^{\kappa_2} - 1 - B_2]$, as required.

Algorithm 5.4 `range_check`: verify that a lies in a signed range using digit decomposition

```
Require: $\bar{a}$, $\bar{B}_1$, $\bar{B}_2$: least residues of $a$, $B_1$, $B_2$
Require: $b_1, b_2, \kappa_1, \kappa_2$: base and digit-length parameters
Require: $LOOKUP_1$, $LOOKUP_2$: lookup tables for digit validity
 1: $x_1 \leftarrow \bar{B}_1 - \bar{a}$
 2: $x_2 \leftarrow \bar{B}_2 + \bar{a}$
 3: $d_1 \leftarrow$ to_base_b$(x_1, b_1, \kappa_1)$
 4: from_base_b$(d_1, x_1, b_1, \kappa_1, LOOKUP_1)$
 5: $d_2 \leftarrow$ to_base_b$(x_2, b_2, \kappa_2)$
 6: from_base_b$(d_2, x_2, b_2, \kappa_2, LOOKUP_2)$
```

```rust
fn range_check<C: Config, API: RootAPI<C>>(
    api: &mut API,
    a: Variable,
    B1_bar: Variable,
    B2_bar: Variable,
    b1: u32,
    kappa1: usize,
    lookup1: &mut LogUpSingleKeyTable,
    b2: u32,
    kappa2: usize,
    lookup2: &mut LogUpSingleKeyTable,
) {
    let x1 = api.sub(B1_bar, a);
    let x2 = api.add(B2_bar, a);

    let d1 = to_base_b(api, x1, b1, kappa1 as u32);
    from_base_b(api, &d1, kappa1, b1, lookup1, x1);

    let d2 = to_base_b(api, x2, b2, kappa2 as u32);
    from_base_b(api, &d2, kappa2, b2, lookup2, x2);
}
```

Listing 5: ECC Rust API: range check using base-b digit constraints

5.4. ReLU. We now extend the range check process to compute ReLU(a).

Algorithm 5.5 `relu`: compute ReLU(a) using digit decomposition

```
Require: $\bar{a}$: least residue of $a$
Require: $b, \kappa$: base and number of digits
Require: $B = (b-1)b^{\kappa-1}$: structural assumption
Require: LOOKUP: lookup table for base-$b$ digit validity
Require: upper_bound: Boolean flag to select method
Ensure: $\bar{y} = \text{ReLU}(a)$
 1: if upper_bound then
 2:     $x \leftarrow B - \bar{a}$                          ▷ Enforces $a \leqslant B$
 3: else
 4:     $x \leftarrow B + \bar{a}$                          ▷ Enforces $a \geqslant -B$
 5: end if
 6: $d \leftarrow$ to_base_b$(x, b, \kappa)$
 7: from_base_b$(d, x, b, \kappa,$ LOOKUP$)$
 8: if upper_bound then
 9:     $s \leftarrow \mathbf{1}_{\{d[\kappa-1]<b-1\}}$
10: else
11:     $s \leftarrow \mathbf{1}_{\{d[\kappa-1]=b-1\}}$
12: end if
13: $\bar{y} \leftarrow s \cdot \bar{a}$
14: return $\bar{y}$
```

```rust
// Verifies whether a lies in a certain interval of length b^kappa and computes ReLU(a)
fn relu<C: Config, API: RootAPI<C>>(
    api: &mut API,
    a: Variable,
    kappa: usize,
    b: u32,
    lookup_table: &mut LogUpSingleKeyTable,
    use_upper_bound: bool,
) -> Variable {
    let b_minus_one = api.constant(b - 1);
    let b_to_kappa_minus_one = api.constant(b.pow((kappa - 1) as u32));
    let b_to_kappa = api.constant(b.pow(kappa as u32));
    let shift = api.mul(b_minus_one, b_to_kappa_minus_one);
    let x = if use_upper_bound {
        // B = (b - 1)b^(kappa - 1), x = B - a
        let b_const = api.mul(b_minus_one, b_to_kappa_minus_one);
        api.sub(b_const, a)
    } else {
        // B = (b - 1)b^(kappa - 1), x = B + a
        let b_const = api.mul(b_minus_one, b_to_kappa_minus_one);
        api.add(b_const, a)
    };

    // Step: Convert x to base-b representation
    let digits = to_base_b(api, x, kappa, b);

    // Step: Constrain digits and enforce reconstruction
    let _ = from_base_b(api, &digits, kappa, b, lookup_table);

    // Step: Compute sign flag from MSB
    let sign = if use_upper_bound {
        // sign = 1 if digits[kappa - 1] < b - 1
        api.assert_is_different(digits[kappa - 1], b_minus_one)
    } else {
        // sign = 1 if digits[kappa - 1] == b - 1
        api.assert_is_equal(digits[kappa - 1], b_minus_one)
    };

    // Step: ReLU(a) = sign * a
    api.mul(sign, a)
}
```
Listing 6: ECC Rust API: verify whether a lies in a certain interval of length b^κ, then compute $\mathrm{ReLU}(a)$

6. Examples

Example 6.1. We fix the prime modulus $p = 101$ and define the offset $h = (p+1)/2 = 50$. We aim to verify whether an integer

$$a \in \{h-p, \dots, h-1\} = \{-50, \dots, 50\}$$

lies in the subinterval

$$\{-27, \dots, -3\}$$

using an arithmetic circuit over $\mathbb{Z}/p\mathbb{Z}$. This is achieved via the range check method described in Section 2 and Algorithm 5.4.

At first glance, the range $[-27, -3]$ may seem arbitrary. However, the underlying idea is straightforward: suppose we wish to verify that $a \leqslant -3$, and we already know that any *honest* prover will supply a value of a within the interval $[-27, -3]$. Then our procedure suffices—we can ignore the tacit verification of the lower bound, as it is guaranteed by context. In practice, for example, the valid values of a might be known to lie in $[-9, -3]$, in which case it is enough to verify $a \leqslant -3$ over the extended interval $[-27, -3]$.

If instead we wanted to verify that $a \geqslant -9$, we could do so using the complementary approach. See Example 6.2 for details.

Let $R = -3$, $b = 5$, and $\kappa = 2$. Then the left endpoint of the interval is given by

$$R - b^\kappa + 1 = -3 - 25 + 1 = -27.$$

We verify the conditions of Corollary 4.2. First, note that

$$b^\kappa = 25 \leqslant b^\kappa - 1 - R + h = 24 + 3 + 50 = 77 \leqslant p,$$

and also that

$$R = -3 \leqslant (b-1)b^{\kappa-1} = 4 \cdot 5 = 20.$$

Thus, the hypotheses of the corollary are satisfied.

We now compute $\bar{R} = R \bmod p = -3 \bmod 101 = 98$. The values $\bar{R}$, b, and κ are hardcoded as circuit constants. For each $a \in \{-50, \dots, 50\}$, the circuit receives the input $\bar{a} = a \bmod 101$, the least nonnegative residue modulo p. The circuit then computes

$$\overline{R-a} = \bar{R} - \bar{a} \bmod 101,$$

which corresponds to computing $R - a$ over the field $\mathbb{Z}/p\mathbb{Z}$ using least residue representatives.

The result $\overline{R-a}$ is then decomposed using Proposition 4.4 and Algorithm 5.1 to obtain digits d_0, d_1, and a quotient q such that

$$\overline{R-a} = d_0 + 5d_1 + 25q,$$

where $d_0, d_1 \in \{0, 1, 2, 3, 4\}$ and $q \in \mathbb{Z}$. The circuit verifies that d_0 and d_1 are indeed base-5 digits either via a lookup table or via a polynomial constraint of the form

$$d_i(d_i-1)(d_i-2)(d_i-3)(d_i-4) \equiv 0 \bmod 101.$$

```rust
// Verifies whether a lies in [-(b - 1)b^(kappa - 1), b^(kappa - 1)) and computes ReLU(a)
fn range_check<C: Config, API: RootAPI<C>>(
    api: &mut API,
    a: Variable,
    kappa: usize,
    b: u32,
    lookup_table: &mut LogUpSingleKeyTable,
) -> Variable {
    // Step 1: Assume a is already in its least residue form modulo p

    // Step 2: Shift a by (b - 1) * b^(kappa - 1)
    let b_minus_one = api.constant(b - 1);
    let b_to_kappa_minus_one = api.constant(b.pow((kappa - 1) as u32));
    let shift = api.mul(b_minus_one, b_to_kappa_minus_one);
    let a_shifted = api.add(a, shift); // a_shifted = a + (b - 1) * b^(kappa - 1) mod p

    // Step 3: Convert a_shifted to a base-b representation with kappa digits (unconstrained)
    let digits = to_base_b(api, a_shifted, kappa, b);

    // Step 4: Enforce digit constraints and reconstruction using lookup table
    // This internally asserts that the digits reconstruct a_shifted
    let _ = from_base_b(api, &digits, kappa, b, lookup_table);

    // Step 5: Compute the sign_a flag for ReLU(a)
    let sign_a = api.is_equal(digits[kappa - 1], b - 1);

    // Step 6: Return ReLU(a) = sign_a * a
    let relu_of_a = api.mul(sign_a, a); // relu_of_a = sign_a * lr_a
    relu_of_a
}
```

Listing 7: ECC Rust API: verify whether a lies in $[-(b-1)b^{\kappa-1}, b^{\kappa-1})$, then verify $\mathrm{ReLU}(a)$

Finally, the circuit verifies the constraint

$$\overline{R-a} \equiv d_0 + 5d_1 \bmod 101. \tag{6.1}$$

This equality holds if and only if $q = 0$, i.e., if $\overline{R-a} \in [0, 24]$. Therefore, the constraint enforces that

$$a \in \{R-24, \dots, R\} = \{-27, \dots, -3\}.$$

For instance, consider $a = -18$. We compute $\bar{a} = -18 \bmod 101 = 83$, and hence

$$\bar{R} - \bar{a} = 98 - 83 = 15.$$

This value lies within the valid digit reconstruction range $[0, 24]$. Applying base-5 expansion,

$$15 = 0 + 5 \cdot 3 + 25 \cdot 0,$$

so $(d_0, d_1, q) = (0, 3, 0)$. Reconstructing $\overline{R-a}$ from the digits gives

$$d_0 + 5d_1 = 15 \equiv 15 \bmod 101,$$

and the constraint (6.1) holds. Thus, the circuit confirms that $a = -18 \in [-27, -3]$ passes the range check.

Next, consider $a = 22$. Here we compute $\bar{a} = 22$, so

$$\bar{R} - \bar{a} = 98 - 22 = 76.$$

This value lies outside the valid reconstruction range. Using base-5 expansion,

$$76 = 1 \cdot 1 + 5 \cdot 0 + 25 \cdot 3,$$

giving $(d_0, d_1, q) = (1, 0, 3)$. Reconstructing using just $d_0 + 5d_1 = 1$ yields

$$76 \not\equiv 1 \bmod 101,$$

so the constraint (6.1) fails. Hence, $a = 22 \notin [-27, -3]$ and is correctly rejected by the range check.

a	$\bar{a}$	$\bar{R}-\bar{a}$ mod 101	(d_0,d_1,q)	d_0+5d_1	(6.1) holds?
-50	51	47	(2,4,1)	22	✗
-49	52	46	(1,4,1)	21	✗
-48	53	45	(0,4,1)	20	✗
-47	54	44	(4,3,1)	19	✗
-46	55	43	(3,3,1)	18	✗
-45	56	42	(2,3,1)	17	✗
-44	57	41	(1,3,1)	16	✗
-43	58	40	(0,3,1)	15	✗
-42	59	39	(4,2,1)	14	✗
-41	60	38	(3,2,1)	13	✗
-40	61	37	(2,2,1)	12	✗
-39	62	36	(1,2,1)	11	✗
-38	63	35	(0,2,1)	10	✗
-37	64	34	(4,1,1)	9	✗
-36	65	33	(3,1,1)	8	✗
-35	66	32	(2,1,1)	7	✗
-34	67	31	(1,1,1)	6	✗
-33	68	30	(0,1,1)	5	✗
-32	69	29	(4,0,1)	4	✗
-31	70	28	(3,0,1)	3	✗
-30	71	27	(2,0,1)	2	✗
-29	72	26	(1,0,1)	1	✗
-28	73	25	(0,0,1)	0	✗

Table 1: Range check analysis for $a \in [-50, -28]$

a	$\bar{a}$	$\bar{R}-\bar{a}$ mod 101	(d_0,d_1,q)	d_0+5d_1	(6.1) holds?
-27	74	24	(4,4,0)	24	✓
-26	75	23	(3,4,0)	23	✓
-25	76	22	(2,4,0)	22	✓
-24	77	21	(1,4,0)	21	✓
-23	78	20	(0,4,0)	20	✓
-22	79	19	(4,3,0)	19	✓
-21	80	18	(3,3,0)	18	✓
-20	81	17	(2,3,0)	17	✓
-19	82	16	(1,3,0)	16	✓
-18	83	15	(0,3,0)	15	✓
-17	84	14	(4,2,0)	14	✓
-16	85	13	(3,2,0)	13	✓
-15	86	12	(2,2,0)	12	✓
-14	87	11	(1,2,0)	11	✓
-13	88	10	(0,2,0)	10	✓
-12	89	9	(4,1,0)	9	✓
-11	90	8	(3,1,0)	8	✓
-10	91	7	(2,1,0)	7	✓
-9	92	6	(1,1,0)	6	✓
-8	93	5	(0,1,0)	5	✓
-7	94	4	(4,0,0)	4	✓
-6	95	3	(3,0,0)	3	✓
-5	96	2	(2,0,0)	2	✓
-4	97	1	(1,0,0)	1	✓
-3	98	0	(0,0,0)	0	✓

Table 2: Range check analysis for $a \in [-27, -3]$

a	$\bar{a}$	$\bar{R}-\bar{a}$ mod 101	(d_0,d_1,q)	d_0+5d_1	(6.1) holds?
-2	99	100	(0,0,4)	0	✗
-1	100	99	(4,4,3)	24	✗
0	0	98	(3,4,3)	23	✗
1	1	97	(2,4,3)	22	✗
2	2	96	(1,4,3)	21	✗
3	3	95	(0,4,3)	20	✗
4	4	94	(4,3,3)	19	✗
5	5	93	(3,3,3)	18	✗
6	6	92	(2,3,3)	17	✗
7	7	91	(1,3,3)	16	✗
8	8	90	(0,3,3)	15	✗
9	9	89	(4,2,3)	14	✗
10	10	88	(3,2,3)	13	✗
11	11	87	(2,2,3)	12	✗
12	12	86	(1,2,3)	11	✗
13	13	85	(0,2,3)	10	✗
14	14	84	(4,1,3)	9	✗
15	15	83	(3,1,3)	8	✗
16	16	82	(2,1,3)	7	✗
17	17	81	(1,1,3)	6	✗
18	18	80	(0,1,3)	5	✗
19	19	79	(4,0,3)	4	✗
20	20	78	(3,0,3)	3	✗
21	21	77	(2,0,3)	2	✗
22	22	76	(1,0,3)	1	✗
23	23	75	(0,0,3)	0	✗
24	24	74	(4,4,2)	24	✗
25	25	73	(3,4,2)	23	✗
26	26	72	(2,4,2)	22	✗
27	27	71	(1,4,2)	21	✗
28	28	70	(0,4,2)	20	✗
29	29	69	(4,3,2)	19	✗
30	30	68	(3,3,2)	18	✗
31	31	67	(2,3,2)	17	✗
32	32	66	(1,3,2)	16	✗
33	33	65	(0,3,2)	15	✗
34	34	64	(4,2,2)	14	✗
35	35	63	(3,2,2)	13	✗
36	36	62	(2,2,2)	12	✗
37	37	61	(1,2,2)	11	✗
38	38	60	(0,2,2)	10	✗
39	39	59	(4,1,2)	9	✗
40	40	58	(3,1,2)	8	✗
41	41	57	(2,1,2)	7	✗
42	42	56	(1,1,2)	6	✗
43	43	55	(0,1,2)	5	✗
44	44	54	(4,0,2)	4	✗
45	45	53	(3,0,2)	3	✗
46	46	52	(2,0,2)	2	✗
47	47	51	(1,0,2)	1	✗
48	48	50	(0,0,2)	0	✗
49	49	49	(4,4,1)	24	✗
50	50	48	(3,4,1)	23	✗

Table 3: Range check analysis for $a \in [-2, 50]$

■

Example 6.2. We fix the prime modulus $p = 101$ and define the offset $h = (p+1)/2 = 50$. We aim to verify whether an integer

$$a \in \{h-p,\dots,h-1\} = \{-50,\dots,50\}$$

lies in the subinterval

$$\{-9,\dots,0\}$$

using an arithmetic circuit over $\mathbb{Z}/p\mathbb{Z}$. This is achieved via the range check method described in Section 2 and Algorithm 5.4. Let $S = 9$, $b = 10$, and $\kappa = 1$. Then the right endpoint of the interval is given by

$$b^{\kappa} - 1 - 9 = 0.$$

We verify the conditions of Corollary 4.3. First note that

$$b^{\kappa} = 9 \leqslant S + h = 9 + 51 = 60 \leqslant p,$$

and also that

$$S = 9 \leqslant (b-1)b^{\kappa-1} = 9.$$

Thus, the hypotheses of the corollary are satisfied.

We now compute $\bar{S} = S \bmod p = 9 \bmod 101 = 9$. The values $\bar{S}$, b, and κ are hardcoded as circuit constants. For each $a \in \{-50,\dots,50\}$, the circuit receives the input $\bar{a} = a \bmod 101$, the least nonnegative residue modulo p. The circuit then computes

$$\overline{S+a} = \bar{S} + \bar{a} \bmod 101,$$

which corresponds to computing $S + a$ over the field $\mathbb{Z}/p\mathbb{Z}$ using least residue representatives.

The results $\overline{S+a}$ is then decomposed using Proposition 4.4 and Algorithm 5.1 to obtain digit d_0 and a quotient q such that

$$\overline{S+a} = d_0 + 10q,$$

where $d_0 \in \{0,\dots,9\}$ and $q \in \mathbb{Z}$. The circuit verifies that d_0 is indeed base-10 digits either via a lookup table or via a polynomial constraint of the form

$$d_0(d_0-1)\cdots(d_0-9) \equiv 0 \bmod 101.$$

Finally, the circuit verifies the constraint

$$\overline{S+a} \equiv d_0 \bmod 101. \tag{6.2}$$

This equality holds if and only if $q = 0$, i.e., if $\overline{S+a} \in [0,9]$. Therefore, the constraint enforces that

$$a \in \{-S,\dots,9-S\} = \{-9,\dots,0\}.$$

For instance, consider $a = -18$. We compute $\bar{a} = -18 \bmod 101 = 83$, and hence

$$\bar{S} + \bar{a} = 9 + 83 = 92.$$

This value lies outside the valid digit reconstruction range $[0,9]$. Applying base-10 expansion,

$$92 = 2 \cdot 1 + 9 \cdot 10 \cdot 0,$$

so $(d_0,q) = (2,9)$. Reconstructing $\overline{S+a}$ from the first digit gives

$$d_0 = 2 \not\equiv 92 \bmod 101,$$

and the constraint (6.2) fails. Hence, $a = -18 \notin [-9,0]$ and is correctly rejected by the range check.

Next, consider $a = -1$. We compute $\bar{a} = -1 \bmod 101 = 100$, and hence

$$\bar{S} + \bar{a} = 9 + 100 = 8.$$

This value lies within the valid digit reconstruction range $[0,9]$. Applying base-10 expansion,

$$8 = 8 \cdot 1 + 0 \cdot 10 \cdot 0,$$

so $(d_0, q) = (8, 0)$. Reconstructing $\overline{S+a}$ from the first digit gives

$$d_0 = 8 \equiv 8 \bmod 101,$$

and the constraint (6.2) holds. Thus, the circuit confirms that $a = -1 \in [-9, 0]$ passes the range check. Combining this with Example 6.1 would verify that $-9 \leqslant a \leqslant -3$.

a	$\bar{a}$	$\bar{S}+\bar{a}$ mod 101	(d_0, q)	d_0	(6.2) holds?
-50	51	60	$(0,6)$	0	✗
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$
-11	90	99	$(9,9)$	9	✗
-10	91	100	$(0,10)$	0	✗
-9	92	0	$(0,0)$	0	✓
-8	93	1	$(1,0)$	1	✓
-7	94	2	$(2,0)$	2	✓
-6	95	3	$(3,0)$	3	✓
-5	96	4	$(4,0)$	4	✓
-4	97	5	$(5,0)$	5	✓
-3	98	6	$(6,0)$	6	✓
-2	99	7	$(7,0)$	7	✓
-1	100	8	$(8,0)$	8	✓
0	0	9	$(9,0)$	9	✓
1	1	10	$(0,1)$	0	✗
2	2	11	$(1,1)$	1	✗
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$
50	50	59	$(9,5)$	9	✗

Table 4: Verification of the constraint $\overline{S+a} \equiv d_0$ mod 101 for $S = 9$, $b = 10$, and $\kappa = 1$. The constraint holds iff $a \in [-9, 0]$.

■

Example 6.3 (ReLU via binary decomposition)**.** Consider the prime $p = 31$, which is a 5-bit prime ($n = 5$), since

$$2^4 \leqslant 31 < 2^5.$$

We take $b = 2, \kappa = 4$, and $h = (p+1)/2 = 16$, which satisfy $b^\kappa \leqslant h + (b-1)b^{\kappa-1} \leqslant p$. We let a range over the balanced residue interval

$$\{h - p, \ldots, h - 1\} = \{-15, \ldots, 15\},$$

and proceed as follows:

- Compute $\bar{a}$, the least residue of a mod p. *Example:* if $a = -15$, then $\bar{a} = 16$.
- Compute $\bar{a} + 2^{\kappa-1}$. *Example:* $\bar{a} + 2^3 = 16 + 8 = 24$.
- Compute $\overline{a + 2^{\kappa-1}}$, the least residue of $a + 2^{\kappa-1}$ modulo p. *Example:* $\overline{a + 2^{\kappa-1}} = 24$.
- Compute $(d_{\kappa-1}, \ldots, d_0)$, the κ least significant bits of $\overline{a + 2^{\kappa-1}}$. *Example:* $\overline{a + 2^{\kappa-1}} = 24$ has binary representation 11000, so

  $$(d_3, d_2, d_1, d_0) = (1, 0, 0, 0).$$

  Note that we discard the leading bit; only the κ least significant bits are retained.
- Impose the constraints:
  - $d_i(d_i - 1) \equiv 0 \bmod p$ for each i.
  - The reconstruction constraint:

$$a + 2^{\kappa-1} \equiv 2^{\kappa-1} d_{\kappa-1} + \cdots + 2^0 d_0 \bmod p. \qquad (*)$$

Example: The first constraint is satisfied: each $d_i \in \{0,1\}$. But the right-hand side of $(*)$ is:

$$2^3(1) + 2^2(0) + 2^1(0) + 2^0(0) = 8,$$

while $a + 2^{\kappa-1} \equiv 24 \bmod p$. Hence, the constraint $(*)$ fails.

- Compute the least residue of the right-hand side of $(*)$. *Example:* $8 \bmod 31 = 8$.
- Compare this with $\overline{a + 2^{\kappa-1}}$ to check whether $(*)$ is satisfied. *Example:* $\overline{a + 2^{\kappa-1}} = 24 \neq 8$, so the constraint fails. This tells us that $a = -15$ lies outside the valid range $[-2^{\kappa-1}, 2^{\kappa-1}) = [-8, 8)$.
- Compute $d_{\kappa-1} \cdot \bar{a}$, which equals $\mathrm{ReLU}(a)$ when the constraints are satisfied. *Example:* $d_3 = 1$ and $\bar{a} = 16$, so $d_3 \cdot \bar{a} = 16$. But $\mathrm{ReLU}(a) = \max\{-15, 0\} = 0$. We would not expect equality in this instance, because a did not pass the range check.

The table below shows the results for all $a \in [h - p, h)$. When $(*)$ holds and the constraints are satisfied, the output of $d_{\kappa-1}\bar{a}$ correctly recovers $\mathrm{ReLU}(a)$.

a	$\bar{a}$	$\bar{a}+2^{\kappa-1}$	$\overline{a+2^{\kappa-1}}$	κ LSBs of $\overline{a+2^{\kappa-1}}$	RHS($*$)	($*$) holds	$d_{\kappa-1}\bar{a}$
-15	16	24	24	1000	8	✗	16
-14	17	25	25	1001	9	✗	17
-13	18	26	26	1010	10	✗	18
-12	19	27	27	1011	11	✗	19
-11	20	28	28	1100	12	✗	20
-10	21	29	29	1101	13	✗	21
-9	22	30	30	1110	14	✗	22
-8	23	31	0	0000	0	✓	0
-7	24	32	1	0001	1	✓	0
-6	25	33	2	0010	2	✓	0
-5	26	34	3	0011	3	✓	0
-4	27	35	4	0100	4	✓	0
-3	28	36	5	0101	5	✓	0
-2	29	37	6	0110	6	✓	0
-1	30	38	7	0111	7	✓	0
0	0	8	8	1000	8	✓	0
1	1	9	9	1001	9	✓	1
2	2	10	10	1010	10	✓	2
3	3	11	11	1011	11	✓	3
4	4	12	12	1100	12	✓	4
5	5	13	13	1101	13	✓	5
6	6	14	14	1110	14	✓	6
7	7	15	15	1111	15	✓	7
8	8	16	16	0000	0	✗	0
9	9	17	17	0001	1	✗	0
10	10	18	18	0010	2	✗	0
11	11	19	19	0011	3	✗	0
12	12	20	20	0100	4	✗	0
13	13	21	21	0101	5	✗	0
14	14	22	22	0110	6	✗	0
15	15	23	23	0111	7	✗	0

Table 5: Range check and ReLU verification for $p = 31$, $b = 2$, and $\kappa = 4$.

We also include two additional values outside the assumed range of a. When $a \notin \{h-p, \dots, h-1\}$, the constraints may fail to hold, revealing that a lies outside the valid range. Even if the constraints are satisfied, the final value $d_{\kappa-1}\bar{a}$ may no longer equal ReLU(a), since the sign bit no longer reliably encodes the correct comparison.

a	$\bar{a}$	$\bar{a}+2^{\kappa-1}$	$\overline{a+2^{\kappa-1}}$	κ LSBs of $\overline{a+2^{\kappa-1}}$	RHS$(*)$	$(*)$ holds	$d_{\kappa-1}\bar{a}$
-30	1	9	9	1001	9	✓	1
16	16	24	24	1000	8	✗	16

Table 6: Examples where $a \notin \{h-p,\dots,h-1\}$: constraints may not be satisfied, or they may be while $\mathrm{ReLU}(a) \neq d_{\kappa-1}\bar{a}$.

■

Example 6.4 (ReLU via base-3 decomposition)**.** Consider the prime $p = 37$, base $b = 3$, $\kappa = 3$, $h = (p+1)/2 = 19$, which satisfy $b^\kappa \leqslant h + (b-1)b^{\kappa-1} \leqslant p$. We let a range over the balanced residue interval

$$\{h-p,\dots,h-1\} = \{-18,\dots,18\},$$

and proceed as follows:

- Compute $\bar{a}$, the least residue of a mod p. *Example:* if $a = -18$, then $\bar{a} = 19$.
- Compute the shifted value
$$\bar{a} + (b-1)b^{\kappa-1} = \bar{a} + 2\cdot 3^2 = \bar{a} + 18.$$
Example: $\bar{a} + 18 = 19 + 18 = 37$.
- Compute $\overline{a+18}$, the least residue of $\bar{a}+18$ mod p. *Example:* $\overline{a+18} = 37 \bmod 37 = 0$.
- Compute the κ least significant base-3 digits of $\overline{a+18}$, denoted (d_2, d_1, d_0). *Example:* $\overline{a+18} = 0$ has ternary representation $(0,0,0)$.
- Impose the constraints:
  - For each i, require
  $$d_i(d_i-1)(d_i-2) \equiv 0 \bmod p,$$
  ensuring $d_i \in \{0,1,2\}$.
  - Enforce the reconstruction constraint
  $$\bar{a} + 18 \equiv 3^2 d_2 + 3^1 d_1 + 3^0 d_0 \bmod p. \qquad (*)$$

  Example: RHS$(*) = 3^2\cdot 0 + 3^1 \cdot 0 + 3^0 \cdot 0 = 0$. Since $\bar{a}+18 = 37$, which is congruent to 0 mod 37, the constraint $(*)$ is satisfied.
- Compute the least residue of the right-hand side of $(*)$. *Example:* $0 \bmod 37 = 0$.
- Compare with $\overline{a+18}$ to determine whether $(*)$ is satisfied. *Example:* $\overline{a+18} = 0$ and RHS$(*) = 0$, so the constraint holds. This confirms that $a = -18$ lies in the valid range $[-18, 9)$.
- Compute the sign flag
$$\mathrm{sign}(a) = \begin{cases} 1 & \text{if } d_{\kappa-1} = b-1, \\ 0 & \text{otherwise.} \end{cases}$$
Example: $d_2 = 0 \neq 2$, so $\mathrm{sign}(a) = 0$.
- Finally, compute
$$\mathrm{ReLU}(a) = \mathrm{sign}(a)\cdot\bar{a}.$$
Example: $0 \cdot 19 = 0 = \mathrm{ReLU}(-18)$, as expected.

The table below shows values for all $a \in [h-p, h)$. For each value, it displays whether the constraint $(*)$ is satisfied, and whether the computed value $\mathrm{sign}(a)\cdot\bar{a}$ matches $\mathrm{ReLU}(a)$.

a	$\bar{a}$	$\bar{a}+18$	$\overline{a+18}$	κ LSDs of $\overline{a+18}$	RHS$(*)$	$(*)$ holds	sign(a)	sign$(a)\bar{a}$
−18	19	37	0	000	0	✓	0	0
−17	20	38	1	001	1	✓	0	0
−16	21	39	2	002	2	✓	0	0
−15	22	40	3	010	3	✓	0	0
−14	23	41	4	011	4	✓	0	0
−13	24	42	5	012	5	✓	0	0
−12	25	43	6	020	6	✓	0	0
−11	26	44	7	021	7	✓	0	0
−10	27	45	8	022	8	✓	0	0
−9	28	46	9	100	9	✓	0	0
−8	29	47	10	101	10	✓	0	0
−7	30	48	11	102	11	✓	0	0
−6	31	49	12	110	12	✓	0	0
−5	32	50	13	111	13	✓	0	0
−4	33	51	14	112	14	✓	0	0
−3	34	52	15	120	15	✓	0	0
−2	35	53	16	121	16	✓	0	0
−1	36	54	17	122	17	✓	0	0
0	0	18	18	200	18	✓	1	0
1	1	19	19	201	19	✓	1	1
2	2	20	20	202	20	✓	1	2
3	3	21	21	210	21	✓	1	3
4	4	22	22	211	22	✓	1	4
5	5	23	23	212	23	✓	1	5
6	6	24	24	220	24	✓	1	6
7	7	25	25	221	25	✓	1	7
8	8	26	26	222	26	✓	1	8
9	9	27	27	000	0	✗	0	0
10	10	28	28	001	1	✗	0	0
11	11	29	29	002	2	✗	0	0
12	12	30	30	010	3	✗	0	0
13	13	31	31	011	4	✗	0	0
14	14	32	32	012	5	✗	0	0
15	15	33	33	100	6	✗	0	0
16	16	34	34	101	7	✗	0	0
17	17	35	35	102	8	✗	0	0
18	18	36	36	110	9	✗	0	0

Table 7: Range check and ReLU verification for $p = 37$, $b = 3$, and $\kappa = 3$.

As with the binary case, if the assumption $h - p \leqslant a < h$ does not hold, the result is not guaranteed to be correct. For example, even if the constraints are satisfied, the computed value $\text{sign}(a) \cdot \bar{a}$ may not equal $\text{ReLU}(a)$.

a	$\bar{a}$	$\bar{a}+18$	$\overline{a+18}$	κ LSDs of $\overline{a+18}$	RHS$(*)$	$(*)$ holds	sign(a)	sign$(a)\bar{a}$
-36	1	19	19	**2**01	19	✓	1	1
19	19	37	0	**0**00	0	✓	0	0

Table 8: Examples where $a \notin \{h-p, \ldots, h-1\}$: constraints may not be satisfied, or they may be while $\text{ReLU}(a) \neq \text{sign}(a) \cdot \bar{a}$.

■

References

[1] Polyhedra Network. *Expander: Proof Backend for Layered Circuits*. `https://github.com/PolyhedraZK/Expander`. Accessed June 6, 2025.

[2] Polyhedra Network. *Expander: Proof System Documentation*. `https://docs.polyhedra.network/expander/`. Accessed June 6, 2025.

[3] Polyhedra Network. *ExpanderCompilerCollection: High-Level Circuit Compiler for the Expander Proof System*. `https://github.com/PolyhedraZK/ExpanderCompilerCollection`. Accessed June 6, 2025.

[4] Polyhedra Network. *ExpanderCompilerCollection: Rust Frontend Introduction*. `https://docs.polyhedra.network/expander/rust/intro`. Accessed June 6, 2025.

[5] Polyhedra Network. *LogUp: Lookup-Based Range Proofs in ExpanderCompilerCollection*. `https://docs.polyhedra.network/expander/std/logup`. Accessed June 6, 2025.