Code Explanation

```
for i in 0..bit_count {
    let a_bit = (a >> i) & T::from(1);
    let b_bit = (b >> i) & T::from(1);

    let sum_bit = a_bit ^ b_bit ^ c_in;
    carry = (a_bit & b_bit) | (b_bit & c_in) | (c_in & a_bit);

    sum |= sum_bit << i;
    c_in = carry;
}
```

Step-by-Step Breakdown

1. Loop Initialization

The loop runs from $i = 0$ to $i <$ bit_count, iterating over each bit position.

2. Extracting Bits

- `let a_bit = (a >> i) & T::from(1);`
  - $(a >> i)$ shifts the bits of a to the right by i positions.
  - $\&T :: from(1)$ isolates the least significant bit (LSB) after the shift, effectively extracting the bit at position i from a.
- `let b_bit = (b >> i) & T::from(1);`
  - Similarly, this extracts the bit at position i from b.

3. Calculating the Sum Bit

- `let sum_bit = a_bit ⊕ b_bit ⊕ c_in;`
  - $\oplus$ is the bitwise XOR operator.
  - The sum bit is calculated using the XOR of a_bit, b_bit, and the carry-in (c_in). This is because XOR of two bits gives the sum without carry.

4. Calculating the Carry

- `carry = (a_bit ∧ b_bit) | (b_bit ∧ c_in) | (c_in ∧ a_bit);`
  - $\wedge$ is the bitwise AND operator.
  - $|$ is the bitwise OR operator.

- The carry is calculated using the AND of pairs of bits and the carry-in. This ensures that the carry is set if any two of the three bits (a_bit, b_bit, c_in) are 1.

5. Updating the Sum

- `sum |= sum_bit << i;`
  - `sum_bit << i` shifts the sum bit to the correct position.
  - $|=$ is the bitwise OR assignment operator, which updates the `sum` by setting the bit at position i to `sum_bit`.

6. Updating the Carry-In

- `c_in = carry;`
  - The carry-out from the current bit position becomes the carry-in for the next bit position.

Example

Let's use some example values to illustrate:

- $a\_bit = 1$
- $b\_bit = 0$
- $c\_in = 1$

For the XOR operation:

1. $a\_bit \oplus b\_bit$ results in $1 \oplus 0 = 1$.
2. $1 \oplus c\_in$ results in $1 \oplus 1 = 0$.

For the OR operation:

1. $(a\_bit \wedge b\_bit)$ results in $1 \wedge 0 = 0$.
2. $(b\_bit \wedge c\_in)$ results in $0 \wedge 1 = 0$.
3. $(c\_in \wedge a\_bit)$ results in $1 \wedge 1 = 1$.
4. Combining these with OR: $0|0|1 = 1$.