创新创业 project3

一、整体结构与设计目标

代码实现了符合参数 (n,t,d)=(256,3,5) 的 Poseidon2 哈希电路，核心目标是：

- 私有输入为哈希原象（一个 block），公开输出为哈希值；
- 支持 Groth16 算法生成零知识证明，验证原象与哈希值的对应关系。
- n=256：输出哈希值为 256 位；
- t=3：算法状态大小为 3（即内部状态由 3 个元素组成）；
- d=5：使用 5 次幂 S-box 作为非线性变换。

整体结构遵循 Circom 的“模板（template）+ 组件（component）”模式，分为：参数配置、置换核心、S-box 实现、哈希函数封装、主电路五个部分。

二、核心组件解析

1. 引用与依赖

```
include "circomlib/circuits/poseidon.circom";
include "circomlib/circuits/utils.circom";
```

引用 circomlib 中的 Poseidon 基础电路和工具函数，便于复用基础组件。

2. Poseidon2 参数配置模板 Poseidon2Params

该模板实现了 Poseidon2 的核心置换（Permutation）过程，是哈希算法的“引擎”。Poseidon 类算法基于**海绵结构**，置换是核心操作（将输入状态通过一系列变换映射到输出状态）。

2.1 轮数配置

```
const ROUNDS_F = 8;        // 全轮数
const ROUNDS_P = 34;       // 部分轮数
const t = 3;               // 状态大小
const d = 5;               // S-box次数
```

- 根据 Poseidon2 论文 Table 1，(256,3,5)参数对应的总轮数为 ROUNDS_F + ROUNDS_P = 42，其中：
  - 全轮（Full Rounds）：前 4 轮 + 后 4 轮，共 8 轮，每轮对**所有状态元素**应用 S-box；
  - 部分轮（Partial Rounds）：中间 34 轮，每轮仅对**第一个状态元素**应用 S-box（减少计算量，保持安全性）。

2.2 轮常量与线性层矩阵

- **轮常量（C）：**每轮添加到状态中的随机常量，用于打破对称性，增强安全性。实际应用中需从 Poseidon2 官方规范获取完整常量（代码中为简化示例，使用全 0）。

```
const C[ROUNDS_F + ROUNDS_P][t] = [
    [0, 0, 0], [0, 0, 0], [0, 0, 0], [0, 0, 0],
    [0, 0, 0], [0, 0, 0], [0, 0, 0], [0, 0, 0],
    // 部分轮常量（仅示例，实际需替换）
    [0, 0, 0], [0, 0, 0], [0, 0, 0], [0, 0, 0],
    [0, 0, 0], [0, 0, 0], [0, 0, 0], [0, 0, 0],
    [0, 0, 0], [0, 0, 0], [0, 0, 0], [0, 0, 0],
    [0, 0, 0], [0, 0, 0], [0, 0, 0], [0, 0, 0],
    [0, 0, 0], [0, 0, 0], [0, 0, 0], [0, 0, 0],
    [0, 0, 0], [0, 0, 0], [0, 0, 0], [0, 0, 0],
    [0, 0, 0], [0, 0, 0], [0, 0, 0], [0, 0, 0],
    [0, 0, 0], [0, 0, 0], [0, 0, 0], [0, 0, 0],
    [0, 0, 0], [0, 0, 0], [0, 0, 0], [0, 0, 0],
    [0, 0, 0], [0, 0, 0]
];
```

- **线性层矩阵（M）：**每轮置换中用于状态混合的线性变换矩阵，需满足可逆性（确保置换是双射）。代码中使用单位矩阵作为示例，实际需用规范定义的矩阵。

```
const M[t][t] = [
    [1, 0, 0],
    [0, 1, 0],
    [0, 0, 1]
];
```

2.3 置换流程

Poseidon2 的置换过程分为三个阶段：**前 4 轮全轮 → 34 轮部分轮 → 后 4 轮全轮**，每轮包含三步：**加常量（AddRoundConstants）→ S-box 变换（SubWords）→ 线性层（MixLayer）**。

- 首先将输入状态（state）复制到临时变量 s，用于后续变换。

```
var s[3];
for (var i = 0; i < t; i++) {
    s[i] = state[i];
}
```

（1） 前 4 轮全轮

- **加常量**：每个状态元素加上对应轮的常量，模 BN254 曲线的阶（p=2188824287...，零知识证明中常用的有限域）。

```
for (var i = 0; i < t; i++) {
    s[i] += C[r][i];
    s[i] = s[i] % 21888242871839275222246405745257275088548364400416034343698204186575808495617;
}
```

- **S-box 变换**：通过 Pow5 函数对每个元素做 5 次幂（x^5 mod p），引入非线性特性（密码学安全性的核心）。

```
// 应用S-box到所有元素（全轮）
for (var i = 0; i < t; i++) {
    s[i] = Pow5(s[i]);
}
```

- **线性层**：通过矩阵乘法混合状态元素，扩散输入信息（确保输入的微小变化影响输出的多个元素）。

```
var new_s[t];
for (var i = 0; i < t; i++) {
    new_s[i] = 0;
    for (var j = 0; j < t; j++) {
        new_s[i] += M[i][j] * s[j];
        new_s[i] = new_s[i] % 21888242871839275222246405745257275088548364400416034343698204186575808495617;
    }
}
```

（2）34 轮部分轮

- 与全轮的唯一区别：S-box 仅作用于第一个状态元素（s[0]），减少计算量（34 轮占总轮数的 81%，此优化可显著降低电路约束数）。

```
for (r = ROUNDS_F/2; r < ROUNDS_F/2 + ROUNDS_P; r++) {
    // 添加轮常量
    for (var i = 0; i < t; i++) {
        s[i] += C[r][i];
        s[i] = s[i] % 21888242871839275222246405745257275088548364400416034343698204186575808495617;
    }

    // 应用S-box到第一个元素 (部分轮)
    s[0] = Pow5(s[0]);
```

（3）后 4 轮全轮

与前 4 轮全轮逻辑完全一致，确保状态经过充分混合，提升安全性。

```
for (r = ROUNDS_F/2 + ROUNDS_P; r < ROUNDS_F + ROUNDS_P; r++) {
    // 添加轮常量
    for (var i = 0; i < t; i++) {
        s[i] += C[r][i];
        s[i] = s[i] % 21888242871839275222246405745257275088548364400416034343698204186575808495617;
    }

    // 应用S-box到所有元素 (全轮)
    for (var i = 0; i < t; i++) {
        s[i] = Pow5(s[i]);
    }

    // 线性层
    var new_s[t];
    for (var i = 0; i < t; i++) {
        new_s[i] = 0;
        for (var j = 0; j < t; j++) {
            new_s[i] += M[i][j] * s[j];
            new_s[i] = new_s[i] % 21888242871839275222246405745257275088548364400416034343698204186575808495617;
        }
    }
```

2.4 输出置换结果

```
for (var i = 0; i < t; i++) {
    out[i] = s[i];
}
```

- 置换结束后，将最终状态 s 作为输出 out。

3. 5 次幂 S-box 函数 Pow5

```
function Pow5(a) {
    var res = a * a;
    res = res * a;  // a^3
    res = res * a;  // a^4
    res = res * a;  // a^5
    return res;
}
```

- S-box（Substitution Box）是密码算法中的非线性变换，用于混淆输入信息。Poseidon2 选用 5 次幂（x^5）是因为：
    - 计算简单（仅需 4 次乘法），电路约束少（适合零知识证明）；
    - 在有限域中是双射（可逆），满足置换的可逆性要求。

4. 哈希函数模板 Poseidon2Hash

该模板实现完整的哈希功能，将输入原象映射为哈希值，基于海绵结构（Sponge Construction）：

```
template Poseidon2Hash() {
    signal private input preimage[2];  // 原象，t=3时rate=2
    signal public output hash[1];      // 哈希结果，取状态的第一个元素作为输出

    // 初始化状态: [0, preimage[0], preimage[1]]
    signal state[3];
    state[0] = 0;
    state[1] = preimage[0];
    state[2] = preimage[1];

    // 应用Poseidon2置换
    component perm = Poseidon2Params();
    for (var i = 0; i < 3; i++) {
        perm.state[i] <== state[i];
    }

    // 输出哈希结果
    hash[0] <== perm.out[0];
```

- **海绵结构解析**：

吸收阶段（Absorb）：将输入原象（preimage）填充到状态中。对于 t=3 的状态，容量（Capacity）为 1 个元素（state[0]），输入长度（Rate）为 t-1=2（即一次可处理 2 个元素），符合代码中 preimage[2]的设计。

置换阶段（Permute）：调用 Poseidon2Params 组件对状态执行置换。

挤压阶段（Squeeze）：从置换后的状态中提取哈希结果（此处取 state[0]作为 256 位输出）。

5. 主电路 main

```
component main { public [hash] } = Poseidon2Hash();
```

- 定义电路的输入输出接口：
    - public [hash]：指定 hash 为公开输入；
    - 私有输入为 Poseidon2Hash 中的 preimage[2]。
- 功能：验证“preimage 经过 Poseidon2 哈希后等于 hash”，即 hash =

Poseidon2(preimage)。

三、电路与零知识证明的适配

该电路专为 Groth16 证明系统设计，满足以下特性：

1. **约束友好**：S-box（5 次幂）仅需 4 次乘法约束，线性层为矩阵乘法（低约束数），整体约束数量远低于 SHA-256 等哈希算法，适合零知识证明。
2. **输入划分合理**：私有输入（原象）和公开输入（哈希值）分离，符合零知识证明“证明者不泄露原象，仅证明哈希关系成立”的需求。