

halo2ex 使用说明

• 概述

halo2ex 封装了 zcash halo2 gadgets 的主要电路和接口，包括 ecc, poseidon, sinsemilla 等，并开发了集成电路以实现 halo2 的电路集成，同时提供了配置接口来生成各种电路以简化开发。

• 单元测试

可在 halo2ex crate 目录下 cargo test 运行单元测试，测试代码在 halo2ex/src/test/目录下，也是使用的示例代码，包括了 sinsemilla short commit, commit, 和集成电路的测试（测试电路类似于 zcash orchard 的 ivk, note commit 和主要的 circuit）

注：本文的测试示例代码都在 halo2ex/src/test/ic/circuit.rs

• 编译和运行

halo2ex 可编译成 halo2ex.rlib Rust 库，也可直接在调用者通过 dependencies 引入（使用本地路径或相对路径引入）。

注：同时需要在调用者的 Cargo.toml 引入：

```
[dev-dependencies]
```

```
halo2_gadgets = { version = "=0.1.0-beta.1", features = ["test-dependencies"] }
```

另外提供了电路调试模式。生成可执行文件时使用如下命令：

```
cargo run --features="debug"
```

可在调用者 crate 目录下生成描述电路结构的 txt 文件（ICCircuit.txt 和 SinsemillaCircuit.txt），以查看配置的电路结构是否正确。同时需要在调用者的 Cargo.toml 引入：

```
[features]
```

```
debug = ["halo2ex/debug"]
```

• 准备工作

* 预生成常量数据（生成元，Running Sum 的 ZS 和 US）

· Sinsemilla

通过: halo2ex::base::print_sinsemilla_commit_domains(name: &str, num_window: usize) 打印出 domain name 对应的 generator (R 点值), zs 和 us

· Merkle 和 Value Commit

通过: halo2ex::base::print_q_hash_domain(name: &str) 打印 Q 点

· Elliptic Curve:

通过: halo2ex::base::print_hash_point_domains(name: &str, num_window: usize,

h: &[u8; 1]) 打印。用于椭圆曲线的快速乘法运算。其中h在zcash orchar中使用了b"K", b"G", b"r", b"v"。

注: num_window需要设置成:

halo2ex::halo2_gadgets::ecc::chip::constants::NUM_WINDOWS或 NUM_WINDOWS_SHORT
以生成full commit或 short commit对应的数据

对于打印的结果数据填入以下结构作为常量:

Full Commit:

GENERATOR: ([u8; 32], [u8; 32])

Z: [u64; NUM_WINDOWS]

U: [[[u8; 32]; H]; NUM_WINDOWS]

Short Commit:

Z_SHORT: [u64; NUM_WINDOWS_SHORT]

U_SHORT: [[[u8; 32]; H]; NUM_WINDOWS_SHORT]

其中 H 是 halo2_gadgets::ecc::chip::H。

* 配置预生成的常量数据

halo2ex::global 里面提供了配置常量数据的全局方法:

- config_generator_q
- config_generator_r
- config_zs_and_us
- config_zs_and_us_short

以下是对应的 gadgets 常量配置:

- config_fixedbasefull
- config_fixedpointbasefield
- config_fixedpointshort

使用上述打印生成的常量对应填入 (其中 name 作为 key 值) .

注: 对于 MerkleCRH 所需的 Q 点 generator, 除了config_generator_q 以外, 还需要调用 global::config_domain_name 做配置, domain name 是 halo2ex::consts::DOMAIN_MERKLECRH, value是config_generator_q 配置的 name.

• 电路配置项

* gate 的配置项

△ 列类型:

- Constant
- Selector
- Fixed
- Advice
- Instance

和 halo2 语义一致

△ 行类型:

- Cur:当前行
- Next:下一行
- Prev:前一行

△ Cell (单元格) 类型:

- Input: 输入cell, 即field element
- YInput:带有y坐标检查的Input
- Instance:存放公用数据的cell
- Piece: field element分解后的整个分片表示, 比如a, b, c, d, e, f, g, 必须是10的整数倍
- Slice: Piece的分解单元, 比如 $b = b_0 || b_1 || b_2$
- TopSlice: slice作为Input的最高位, 必须是1个字节
- PadSlice:当Piece分解为slice不足以填充Input时候的填充位, 一般是整个分解的最后的位
- YSlice:作为Input y坐标的slice, 一般是1个字节
- CanonicityCheckSlice:作为canonicity check的cell, 同时是slice
- CanonicityCheckZ13: z13的canonicity check的cell, 不能同时是slice
- CanonicityCheck: canonicity check的cell, 不能同时是slice
- PrimeCheck: prime check的cell

* 运算类型/操作符

- add: 加法
- sub: 减法
- mul: 乘法
- boolean_neg: 布尔变量的非运算, 即 $0 \rightarrow 1, 1 \rightarrow 0$, 仅用于 gate 约束运算
- poseidon: 仅用于组合运算

* 变量类型

- Field: 即 pallas::Base 类型, synthesize 的时候通过 private_load 载入
- Point: 即 pallas::Point 类型, synthesize 的时候转换成 ecc::Point
- NIPoint: 即 pallas::Point 类型, synthesize 的时候转换成 ecc::NonIdentityPoint
- Scalar: 即 pallas::Scalar 类型, 一般作为随机因子
- Cell: 单元格类型, 即 halo2_proofs::circuit::AssignedCell, 一般是 synthesize 时作运算后的变量, 不作为外部输入
- CommitCell: sinsemilla commit 的结果类型, 也是 AssignedCell
- MagnitudeSign: Value 减法的 witness 变量类型 (参见测试代码中的 magnitude_v 和 sign_v)
- FullField: 即 ecc::FixedPoint, 一般用于 ecc 乘法运算, 需要预生成的常量数据配置
- BaseField: 即 ecc::FixedPointBaseField, 一般用于 ecc 乘法运算, 需要预生成的常量数据配置
- ShortField: 即 ecc::FixedPointShort, 一般用于 ecc 乘法运算, 需要预生成的常量数据配置

• 配置开发

需要实现电路接口的 traits (定义在 halo2ex::circuit::base) :

```
pub trait InstanceOrder {
    fn get_instance_order() -> Vec<String>;
}

和

pub trait ICConfig: InstanceOrder {
    type Value: Signed;
    fn get_ic_configs() -> (Vec<GateConfig>, Vec<Vec<AlgoConfig>>);
    fn get_commit_gate_configs(domain: &String) -> Option<Vec<GateConfig>>;
    fn get_commit_configs(
        ) -> Option<Vec<(bool, String, (String, usize), Vec<(String, String)>, String)>>;
}
```

* InstanceOrder

公共输入 name 的配置, 返回 instance name 的列表 Vec。注: 在 synthesize 产生约束的时候会依赖 instance 在 Vec 的顺序。

* ICConfig

主要的配置接口, 包括了以下几个部分 (配置用到的数据定义在 halo2ex::types中) :

• Value Commit

通过配置 Value 类型实现需要检查的 value commit, 类型必须是 i16, i32, i64, i128 中的一项, value 和 value 运算结果的关系是:

u8 -> i16: 即 u8 的 value 计算后得到 i16

u16 -> i32: 即 u16 的 value 计算后得到 i32

u32 -> i64: 即 u32 的 value 计算后得到 i64

u64 -> i126: 即 u64 的 value 计算后得到 i126

外部输入 value 按 unsigned 类型进行初始化, 即上面的 u 部分, 否则会报溢出错误。(i 部分是 ICConfig 的 Value 类型)

注: 目前内置只支持减法的 commit 检查。

• Sinsemilla Commit

需要实现如下 2 个接口:

◆ Commit gate 的配置: get_commit_gate_configs(domain: &String) ->
Option<Vec<GateConfig>>

输入: domain name, 和预生成数据的配置一致, 可配置多个 domain
返回值: GateConfig 的列表配置。GateConfig 的定义如下:

```
pub type GateConfig = (
    String,
    Vec<(String, String, String, String, usize, String, usize)>,
);
```

第一个是 gate 的 name，第二个 Vec 是 cell 的配置：

- 1) cell name
- 2) attribute: cell 的属性，此处忽略，即为""
- 3) cell type: 即上面定义的 Cell 类型
- 4) col type: 列的类型
- 5) col index: 列的索引
- 6) row: 行的索引，即 Cur, Next 和 Prev
- 7) width: 字节宽度

注：

- 配置的第一个 cell 的类型必须是 Input, YInput 或者 Piece，并且 Piece 的配置在 Input 之前
- 第一个 cell 的 width 应该是后面分片的 width 的和（不包括YSlice和running sum相关的项）

◆ Commit输入的配置: `fn get_commit_configs() ->`

```
Option<Vec<(bool, String, (String, usize), Vec<(String, String)>, String)>>;
```

返回值是输入的配置：

- 1) is short commit: 是否是 short commit 或者 full commit
- 2) commit name: commit 计算完成后的变量名，可作为 key 参与后续的运算
- 3) domain name 和 num window: 和配置的预生成数据匹配
- 4) input 的列表，包括 name 和变量类型。如果变量已在运算配置中，则可为空
- 5) random input name: 输入的随机变量 name

• 集成电路配置

需要实现如下接口：

```
fn get_ic_configs() -> (Vec<GateConfig>, Vec<Vec<AlgoConfig>>)
```

第一个返回是 gate 的列表，第二个是 gate 对应的约束运算规则和组合运算的配置。

注：因为每个 gate 需要对应 1 个约束运算规则，所以第一个 Vec 的 len 必须小于等于第二个

◆ gate 的配置：

和 sinsemilla 的基本一致，但需要处理 attribute：

- Value 类型: attribute 填"Value"，cell 的 name 必须以"old_"或者"new_"开头，后面跟 value 变量的 name（参见测试代码中的"v", "old_v", "new_v"）

• Merkle 类型:

- name: 必须以"anchor_"开头（参见测试代码中的 anchor_merklecrh_cm）
- attribute 的格式:

"MerklePath:{Merkle变量的name}#{配置预生成数据的name}#{Merkle输出值的name}"

其中 Merkle 变量的 name 用于标识，不参与运算

- 其他类型: attribute 为空

◆ gate 运算约束的配置:

一个 gate 的约束可以分解为 1 个或多个运算规则（必须和 gate 的数量相同）。运算的定义（在 halo2ex::base）：

```
pub type AlgoConfig = (  
    String,  
    String,  
    Vec<(  
        String,  
        String,  
        String,  
        (String, String),  
        String,  
        Option<(String, String)>,  
    )>,  
);
```

- 1) 运算结果的 name，可以为空
- 2) 运算的描述
- 3) 分解的单个运算的步骤列表，分为：
 - 1) 和前一个运算结果的运算操作符，所以第一项为空
 - 2) 本次单个运算结果的 name，可以参与后续的运算，否则可以为空
 - 3) 本次运算的描述
 - 4) 左操作数：name 和变量类型
 - 5) 运算操作符
 - 6) 右操作数：name 和变量类型，为 None 时直接返回左操作数作为本次的结果

注：一个复杂多步的运算总可以分解成上述单个运算

◆ 组合运算的配置:

和上面相同，但需要配置在上面运算约束的后面。

不同之处：AlgoConfig 的第一项 name 必须是 constraint 或者 constraint-commit，因为组合运算的结果需要参与（和其他运算结果或公共输入的）约束：

- constraint: 2 个或多个变量组合运算的约束
- constraint-commit: sinsemilla commit 运算结果的约束（commit 的运算结果可生成约束也可继续参与其他组合运算）
- 对于 witness 输入的约束，运算结果的变量 name 需要以“constraint”开头（参见测试代码中的 constraint_derived_pk_d_old）

注：

- 组合运算的结果或者中间单个运算的操作数可以是（依赖）其他运算的结果，但必须按照顺序配置（Vec 中的先后顺序）
- 依赖关系不应过于复杂，更不能循环依赖

• 运算的类型规则

根据 halo2_gadgets::ecc 中的定义，目前支持的运算变量（ECC 上的运算）规则如下：

运算	左操作数类型	右操作数类型	结果类型
add	Point	Point 或 NIPoint	Point
	NIPoint	Point 或 NIPoint	Point
	Cell 或 CommitCell	Cell 或 CommitCell	Cell
mul	NIPoint	Cell 或 CommitCell	Point
	Cell 或 CommitCell	Cell 或 CommitCell	Cell
	FullField	Scalar	Point
	BaseField	Cell 或 CommitCell	Point
	ShortField	MagnitudeSign	Point
poseidon	Cell 或 CommitCell	Cell 或 CommitCell	Cell

• 实例化电路

* ICCircuit

实例化对象 `halo2ex::circuit::ic::ICCircuit` 时需特化上面实现的 traits。

ICCircuit 提供了 `add_xxx` 方法来输入外部数据，具体可参考测试代码。

其中：

- `add_values` 需要同时输入 `old value` 和 `new value`，`name` 需要传入实际的 `value name`（不带"old_"和"new_"）
- `add_merkle_data` 需要传入和上面 `MerkleCRH` 的 `attribute` 相同配置的数据，示例参见测试代码
- `add_constraint_point`：传入作为 `witness` 约束的 `point`，参见示例中的 `constraint_derived_pk_d_old`
- 通过 `add_xxx` 传入的各种数据，可按上面的表格类型参与各种运算或生成约束

* Sinsemilla

在 `halo2ex::domains` 中提供了方法：

```
pub fn compute_commit_value(
    is_short_commit: bool,
    commit_domain_name: &str,
    input_r: &pallas::Scalar,
    inputs: &CommitInputs,
) -> CommitResult
```

来计算 `commit` 的值，其中：

- `commit_domain_name` 对应配置预生成数据的 `domain name`
- `input_r` 是输入的随机值
- `inputs` 是输入值的 `Vec`，并可输入对于 `full commit` 的 `y` 坐标值，具体可参见测试代码
- 返回值是一个 `enum`：`short commit` 是 `x` 坐标值，`full commit` 是 `point` 值

* Instance 输入

Instance 对象（公共输入）包含 2 个值对象：

- enables: 用于电路 gate 的布尔值约束
- fields: public 的输入值, 用于约束验证

* poseidon

可通过 `halo2ex::primitives::utils::poseidon_hash` 计算 hash 值

注: 以上输入值均是通过 name 和运算配置的变量匹配

• Proof 和 Verify

- 1) 初始化时通过 `halo2ex::circuit::proof::ProvingKey` 和 `VerifyingKey` 预先生成 pk 和 vk
- 2) 通过 `halo2ex::circuit::proof::Proof::create` 创建 proof
- 3) 通过 `proof.verify` 验证 proof

注:

- * 可同时进行批量的 proof 创建和验证
- * 经实测, 待验证的 halo2 电路越多, 创建的 proof 越大、耗时也越长, 但 verify 的时间基本恒定

• Primitives 及扩展

`halo2ex::primitives` 下提供了一些对基本数据类型的封装 (类似于 zcash orchard) :

- * Commitment
- * 类似于 redpallas 实现的 `ValidatingKey`
- * `NonIdentityPallasPoint`
- * Nullifier
- * `DomainMerkleHash`
- * `ValueCommitment`

可以在其上扩展实现各种运算, 以生成 instance 或约束验证数据 (可参考测试代码的 `Randomizer`, `Nullifier::derive`, `ValueCommitment::derive`)

• 常量说明

- * field element 的宽度: `halo2ex::consts::FILED_SIZE`, 默认是 255
- * merkle tree 的深度: `halo2ex::consts::MERKLE_DEPTH`, 默认是 32
- * K 值: 即 halo2 table 的行数基数, zcash 使用的是 11

• 参考链接

- * zcash halo2: <https://github.com/zcash/halo2>
- * zcash orchard: <https://github.com/zcash/orchard>
- * zcash orchard book: <https://zcash.github.io/orchard/index.html>