# JBUniswapV4Hook - Juicebox × Uniswap V4 Integration

**Official Juicebox integration for Uniswap v4 that provides price comparison and optimal routing with TWAP oracle protection** 🦄 🧃

## Overview

The JBUniswapV4Hook is a Uniswap V4 hook that intelligently routes swaps between Uniswap pools and Juicebox project token minting. It compares prices in real-time and automatically routes to the option that gives users the most tokens, while using a TWAP (Time-Weighted Average Price) oracle to protect against price manipulation and front-running attacks.

### Key Features

- ✅ **Automatic Price Comparison** - Compares Uniswap vs Juicebox prices on every swap
- ✅ **Optimal Routing** - Routes to the cheaper option (more tokens for user)
- ✅ **TWAP Oracle Protection** - Protects against price manipulation and front-running
- ✅ **Multi-Currency Support** - Works with ETH and ERC20 tokens
- ✅ **Event Transparency** - Emits price comparison and routing decision events
- ✅ **Flexible Integration** - Works with any Juicebox project token

## How It Works

### Price Comparison Flow

```
User initiates swap → Hook checks both routes:
├─ Uniswap Route: Uses TWAP oracle for manipulation-resistant pricing
└─ Juicebox Route: Calculates tokens based on project weight & currency rates

→ Compare outputs → Route to option giving MORE tokens → User receives optimal amount
```

### TWAP Oracle Protection

The hook implements a Time-Weighted Average Price oracle that:

- Records price observations after each swap
- Uses historical data to calculate average prices over 30-minute windows
- Makes price manipulation attacks significantly more expensive
- Protects users from front-running by using stable average prices instead of volatile spot prices

**Without TWAP**: Attacker can manipulate spot price → Victim pays inflated price
**With TWAP**: Attacker's manipulation has limited impact → Victim protected by historical average

# Architecture

## Core Components

### `JBUniswapV4Hook.sol`

Main hook contract implementing:

- `beforeSwap()` - Price comparison and routing logic
- `afterSwap()` - Oracle observation recording
- `afterInitialize()` - Oracle initialization for new pools
- TWAP oracle implementation using Uniswap V4's observation library

**Hook Permissions**

```
afterInitialize: true    // Initialize oracle observations
beforeSwap: true         // Compare prices and route
afterSwap: true          // Record oracle observations
beforeSwapReturnDelta: true  // Override swap behavior for Juicebox
routing
```

## Juicebox Integration

The hook integrates with Juicebox protocol contracts:

- **IJBTokens** - Identifies which tokens are Juicebox project tokens
- **IJBMultiTerminal** - Processes payments to mint project tokens
- **IJBController** - Retrieves project weight (tokens per ETH)
- **IJBPrices** - Converts between currencies for accurate comparisons

## State Variables

```
mapping(PoolId => uint256) public projectIdOf;  // Cached project IDs
mapping(address => uint256) public currencyIdOf;  // Token → Currency ID
mapping
mapping(PoolId => Oracle.Observation[65535]) public observations;  //
TWAP data
mapping(PoolId => ObservationState) public states;  // Oracle state per
pool
```

# Testing

## Comprehensive Test Suite

The project includes **33 tests with 100% pass rate**, including:

**Unit Tests (11)**

- Token calculation tests (ETH, multi-currency, edge cases)
- Hook permissions verification
- Project detection and caching
- Oracle initialization
- Currency ID management

**Fuzz Tests (22)**

Advanced property-based tests with 256+ runs each:

**TWAP Oracle Tests:**

- `testFuzz_TWAPFallbackToSpot` - Fallback behavior
- `testFuzz_TWAPBuildupOverTime` - Observation history building
- `testFuzz_TWAPTimeWeighting` - Time-weighted calculations
- `testFuzz_CardinalityImpactOnTWAP` - Cardinality effects

**EdgeCase**

- `testFuzz_PriceManipulationResistance` - Verifies TWAP protects against price manipulation
- `testFuzz_FrontRunningProtection` - Verifies TWAP protects victims from front-running attacks

**Routing & Price Comparison:**

- `testFuzz_RoutingToLowestPrice` - Optimal routing verification
- `testFuzz_PriceComparisonLogging` - Event emission verification
- `testFuzz_ExtremePriceScenarios` - High volatility handling

**Additional Fuzz Tests:**

- Token calculation with various weights and amounts
- Currency conversion with price feeds
- Zero weight and invalid project handling
- Project ID caching across various scenarios

## Test Documentation

All tests include **Gherkin-style documentation** for clarity:

```
/// Given a pool where spot price is manipulated
/// When comparing spot price vs TWAP price
/// Then TWAP should provide price manipulation resistance
function testFuzz_PriceManipulationResistance(...)
```

### Running Tests

```
# Run all tests
forge test

# Run with verbosity to see events
forge test -vvv

# Run specific test suite
forge test --match-contract JuiceboxHookTest

# Run only TWAP oracle tests
forge test --match-test testFuzz_TWAP

# Run security tests (manipulation & front-running)
forge test --match-test
"testFuzz_PriceManipulation|testFuzz_FrontRunning"

# Generate gas report
forge test --gas-report

# Run with coverage
forge coverage
```

### Test Results

```
✅ All 33 tests passing (100%)
✅ 11 unit tests
✅ 22 fuzz tests (256+ runs each)
✅ Zero failures
✅ Coverage: TWAP oracle, routing logic, price calculations, attack
scenarios
```

## Deployment

### Prerequisites

```
forge install
```

### Local Development

1. Start Anvil:

```
anvil
```

2. Deploy the hook:

```
forge script script/DeployJBUniswapV4Hook.s.sol \
    --rpc-url http://localhost:8545 \
    --private-key
0x59c6995e998f97a5a0044966f0945389dc9e86dae88c7a8412f4603b6b78690d \
    --broadcast
```

### Network Deployment

1. Store your private key securely:

```
cast wallet import <KEY_NAME> --interactive
```

2. Deploy:

```
forge script script/DeployJBUniswapV4Hook.s.sol \
    --rpc-url <YOUR_RPC_URL> \
    --account <KEY_NAME> \
    --sender <YOUR_ADDRESS> \
    --broadcast
```

## Configuration

### Setting Currency IDs

For non-ETH tokens, set the Juicebox currency ID:

```
hook.setCurrencyId(tokenAddress, currencyId);
```

### Increasing Oracle Cardinality

For better TWAP precision, increase observation capacity:

```
hook.increaseCardinalityNext(poolId, 100); // Store 100 observations
```

## Security Considerations

### TWAP Oracle

- **30-minute lookback period** - Configurable via `TWAP_PERIOD` constant
- **Minimum 2 observations required** - Falls back to spot price if insufficient data
- **Gradual price updates** - Manipulation requires sustained attacks
- **Cost of attack** - Would need to maintain manipulated price for 30+ minutes

### Price Manipulation Resistance

The TWAP oracle makes manipulation attacks economically unviable:

1. Attacker would need massive capital to move price significantly
2. Must maintain manipulation for entire TWAP period (30 min)
3. Arbitrageurs would profit from the manipulation
4. Other traders would front-run the attacker's exit

### Front-Running Protection

Users are protected because:

- Hook uses TWAP prices, not spot prices
- Single-block manipulation has minimal TWAP impact
- Front-runners can't exploit temporary price spikes
- Victims pay fair time-weighted average prices

## Gas Optimization

The hook is optimized for gas efficiency:

- Project IDs are cached after first detection
- TWAP calculations use efficient Uniswap V4 library
- Only active hooks are executed (before/after swap when needed)
- Minimal storage writes (observations ring buffer)

## Events

```
event JuiceboxPaymentProcessed(
    PoolId indexed poolId,
    address indexed token,
    uint256 indexed projectId,
    uint256 amount,
    uint256 tokensReceived
);

event PriceComparison(
    PoolId indexed poolId,
    uint256 uniswapPrice,
    uint256 juiceboxPrice,
    bool juiceboxCheaper,
    uint256 priceDifference
);
```

```
event RouteSelected(
    PoolId indexed poolId,
    bool useJuicebox,
    uint256 expectedTokens,
    uint256 savings
);
```

## Examples

### Pool with Juicebox Token

```
// Token0 is a Juicebox project token with ID 123
// When users swap Token1 → Token0:
// 1. Hook detects project ID 123
// 2. Calculates Juicebox route: weight × amount / 1e18
// 3. Calculates Uniswap route: TWAP-based estimate
// 4. Routes to option giving more tokens
// 5. Emits PriceComparison and RouteSelected events
```

### Non-Juicebox Pool

```
// Neither token is a Juicebox project
// Hook allows normal Uniswap swap
// No routing or price comparison needed
```

## Troubleshooting

### Hook Deployment Failures

Ensure hook permissions match flags:

```
// In HookMiner.find():
Hooks.AFTER_INITIALIZE_FLAG |
Hooks.BEFORE_SWAP_FLAG |
Hooks.AFTER_SWAP_FLAG |
Hooks.BEFORE_SWAP_RETURNS_DELTA_FLAG
```

### TWAP Returns Zero

Possible causes:

- Pool too new (< 2 observations)
- Insufficient time elapsed (< TWAP_PERIOD)

- No swaps in lookback window

Solution: System automatically falls back to spot price

### Price Comparison Issues

Check:

- Currency IDs are set correctly for all tokens
- Juicebox project has non-zero weight
- Price feed exists for currency conversions

## Contributing

### Development Setup

```
git clone <repo>
cd juicebox-uniswap-v4
forge install
forge test
```

### Testing Guidelines

- All new features must include tests
- Add Gherkin comments to test functions
- Fuzz tests should have reasonable bounds
- Security-critical functions need property-based tests

### Code Style

- Follow Solidity style guide
- Use NatSpec comments for all public functions
- Keep functions focused and single-purpose
- Emit events for important state changes

## Additional Resources

### Juicebox

- Juicebox Docs
- Juicebox Protocol

### Uniswap V4

- Uniswap v4 docs
- v4-periphery
- v4-core
- v4-by-example

Oracle Resources

- Uniswap V3 Oracle Guide
- TWAP Best Practices

## License

MIT

## Security

For security concerns, please contact: security@juicebox.money

**Audits**: Not yet audited - use at your own risk in production