ZeroSlip

Juuso Roinevirta
juuso@ape.llc

January 2023

February 2023 Revision

Abstract
This whitepaper introduces the concept of a zero slippage, zero price impact automated market maker, a zsAMM. It outlines some of the motivations and design decisions behind the intended implementation of the zsAMM in the ZeroSlip core contracts. It gives a high-level overview of the core contracts and suggests potential use cases for the zsAMM beyond implementing equal asset swaps. Finally, the paper suggests potential future versions and modifications.

Contents

0. Definitions

AMM. Automated market maker.

Callable liquidity. A feature of the zsAMM where an LP commits to providing liquidity on demand while retaining the ability to use and commit their assets elsewhere.

CLOB. Central limit order book. A common system for matching buyers and sellers in centralised trading systems.

DEX. Decentralised exchange. Commonly, an AMM implemented on a blockchain.

Equal assets. Equal assets are considered asset pairs whose value is the same or whose price is determined based on the same factors and will always return to equivalency over time. Equal assets may also be priced equal to each other, subject to a ratio or a multiplier. Generally, equal assets will exhibit the property $price_0 \sim k \times price_1$ over the long-term while short-term fluctuations may cause temporary shifts in market price where $price_0 \neq k \times price_1$.

ERC20. A standard and widely used design for fungible tokens within EVM environments.

EVM. Ethereum Virtual Machine.

Invariant. An immutable property. In this paper, it usually refers to the zsAMM invariant *(see Appendix B).*

LP. Liquidity provider. An entity supplying supported assets to the zsAMM pool.

OTC. Over-the-counter. Usually, refers to trading assets via broker-dealer networks directly between two parties in bilateral agreements or through centralised clearing counterparties.

zsAMM. Zero slippage automated market maker. The concept introduced in this paper.

1. Introduction

ZeroSlip is an on-chain system of smart contracts built in Solidity for use on the Ethereum Virtual Machine. It implements a zero slippage automated market maker ("zsAMM") protocol similar to the design of Uniswap v2 with a few changes restricting its scope and uses further from the general purpose solution. While the base use case for the zsAMM is swapping equal assets, its design also introduces special properties & use cases as outlined in section 5, such as CLOB-like properties, binary options, and token migration contracts.

The zsAMM implements a strict invariant condition $token_0 = k \times token_1$ where k is an immutable variable defined at the construction of the contract. Due to the invariant, the prices of the tokens are always fixed to each other at a constant exchange rate. This reduces the potential uses of the zsAMM to a very limited set of assets whose value is the same or whose price is determined based on the same factors and will always return to equivalency over time. In this paper, assets of this type are referred to as *equal assets*. Creating a zsAMM pool for assets whose value is expected to diverge from each other over time will lead to irrecoverable losses to liquidity providers *("LP")* through arbitrage.

While section 4 covers some of the functions exposed by the zsAMM core contracts, the list of functions is not exhaustive, and this paper does not intend to serve as a technical reference. Furthermore, it should be noted that it may be more beneficial for users to access the zsAMM core contracts through routers which are not covered in this paper.

2. Motivation

The current design space of automated market maker ("AMM") systems is mostly for generalised applications. Such AMMs include, among others: a) Uniswap v2, designed for the exchange of two arbitrary assets following the $x \times y = k$ constant product formula (Adams et al., 2020); b) Curve's

StableSwap, designed for the exchange of n assets whose value closely tracks each other by implementing the StableSwap invariant, which can be generalised as $(x + y) + (x \times y) = D + \left(\frac{D}{n}\right)^n$ where n is the number of assets in the pool (Curve.fi, 2021; Mota, 2021); c) Uniswap v3, designed for the exchange of two arbitrary assets, through the introduction of *concentrated liquidity* – or bounded liquidity – (Adams et al., 2021) proxying the properties of a central limit order book *("CLOB")*, given the restrictions of the EVM; and d) Balancer invariant spot price surface.

While all of these designs are highly effective and efficient for different use cases, none of them serves equal asset swaps with high efficiency; with Uniswap v2, traders will incur high costs due to price impact, StableSwap subjects the traders to high gas costs due to the complex calculations required to check the invariant conditions, and Uniswap v3's bounded liquidity is computationally inefficient for LPs and, also, traders if liquidity bounds are crossed.

Hence, the most effective way to access liquidity in equal assets is through a CLOB or an over-the-counter *("OTC")* desk, assuming liquidity is available. Furthermore, providing liquidity to CLOBs or OTC desks is not permissionless, and they do not enjoy the same economies-of-scale benefits as their permissionless alternatives.

ZeroSlip's zsAMM aims to target the niche of trading equal assets. It focuses on providing extremely gas-efficient trades between two assets while subjecting LPs to high trust assumptions with respect to the value of the assets within the trading pair. The gas efficiency of the zsAMM is further explored in Appendix A. Consequently, the LPs in a zsAMM should be extremely conscious of the nature of the assets they are providing, as a discrepancy in the price of one asset will cause the higher-value asset to be completely drained from the pool.

Given the focus of the zsAMM on computationally efficient trades between arbitrary equal assets, most of the volume in a zsAMM system is anticipated to originate from actors such as arbitrage bots, market makers, and other non-human-users accessing the zsAMM for liquidity.

3. Guiding principles

ZeroSlippage's zsAMM is designed to be: a) permissionless – anyone with access to the environments in which zsAMM is deployed in can choose to use it, either as an LP or as a trader; b) computationally efficient – all actions, especially those of the traders, should be computationally as efficient as possible to allow for low-cost transactions and utilisation of the smallest possible arbitrage opportunities to provide market efficiency; c) composable – the design is centred around generality, such that it can be integrated into other protocols, aggregators, and systems; d) generally applicable – the design allows for various equal assets to be traded, regardless of the value of k, and LPs to set risk parameters through the use of various fee schemes; and 3) immutable – once deployed, the core functions of the smart contracts cannot be changed.

4. Mechanism & Features

At its core, zsAMM is a simple peer-to-pool contract that allows two distinct parties to interact with the pool; traders & LPs.

Traders may interact with the pool to swap an asset ("$token_0$") for another asset ("$token_1$") at a predetermined ratio $token_0 = k \times token_1$ *("invariant")*. Swaps work bidirectionally, hence it is possible to also swap $token_1$ for $token_0$ as long as the invariant is not violated. The invariant ensures all trades happen at a pre-defined rate, less fees. If, for example, $k = 1$ a swap from $n\, token_0$ gives the trader exactly $n\, token_1$, less fees. Furthermore, trying to swap $token_0$ for $n\, token_1$ will fail if n exceeds the reserve balance of $token_1$ in the pool and vice versa. Reserve balances of $token_0$ and $token_1$ are denoted $reserve_0$ and $reserve_1$, respectively.

LPs may interact with the pool to exchange assets supported by the pool to liquidity provider tokens ("LPT"). LPT accrue fees paid by traders from swaps happening within the pool through the increase in assets per LPT within the pool. Liquidity of a pool, or the reserve value of the pool, is calculated as $RV = reserve_0 + k \times reserve_1$ and the value of LPT, denominated in $token_0$, is calculated as $LPT = \frac{RV}{Count\ of\ LPT}$ *("LPT value formula")*. Hence, as the pool's reserve value increases due to accrued fees from swaps, the value of an LPT increases through an increased claim to the reserve assets, *ceteris paribus*.

In an effort to improve the capital efficiency of the LPs, the zsAMM also introduces the concept of *callable liquidity*. Callable liquidity allows LPs to commit their assets for use in trades on demand. For example, an LP can commit 100 units of $token_0$ for use in trades in pools x and y, simultaneously, while retaining possession of the tokens. Should a trader want to trade $token_1$ for $token_0$ in pool x, the trader can call the liquidity from the LP, on demand, in order to fulfil their trade. Effectively, the LP is not committing or "locking" any capital prior to the trade taking place, ensuring their capital is directed to the most demanded trading venue at any one time, improving the LP's capital efficiency.

Two fee types apply to interactions with the zsAMM: the swap fee rates (F_S) and LP withdrawal fee rate (F_W). All fee rates are defined in tenths of a basis point ($\frac{1}{100\,000}$).

The swap fee is applied to swaps between two assets. The swap fee rate can be determined based on the direction of the swap; F_{S1} applies to swaps from $token_0$ to $token_1$ and F_{S0} applies to swaps from $token_1$ to $token_0$. The swap fees are charged from the token sent ($token_{in}$) to the zsAMM, maximising liquidity of the outbound token ($token_{out}$) and allowing one side of the pool to be completely drained with a single transaction.

Due to potential fee skirting issues associated with low trade sizes (i.e. cases where $token_{in} \times F_S < 100\,000$), the absolute swap fee is determined as $F_{Sa} = \{F_S = 0 \; ? \; 0 : n\,F_S + 1\}$ where n is the amount of $token_{in}$. Hence, if $F_S > 0$ the minimum absolute swap fee is always 1 unit of $token_{in}$.

The withdrawal fee is applied to LPs burning LPT for one of the pool's tokens. The withdrawal fee rate is defined as $F_W = \left\{F_S = 0 \; ? \; 1 : \frac{F_S}{4}\right\}$ where F_S is the swap fee to $token_{out}$. Similar to the swap fee, the withdrawal fee would be subject to fee skirting issues with withdraw sizes satisfying $token_{in} \times F_S < 400\,000$. Hence the absolute withdrawal fee is defined as $F_{Wa} = n\,F_W + 1$ where n is the amount of $token_{out}$. It follows from the fee structure that LPs can lose money a) by exiting the pool before swap fees have recouped the cost of exit; b) if they provided liquidity to a zero-fee pool; or c) through impermanent loss.

It follows from the properties of the zsAMM that it is a) a *zero slippage* AMM as there is no external price input; and b) a *zero price impact* AMM as k is immutable.

4.1 ZeroSlipFactory

The ZeroSlipFactory is a utility used for the creation of new ZeroSlipPools. It exposes a public function called `createZeroSlipPool()` which allows the permissionless creation of a new zsAMM pool. It takes the addresses of $token_0$ & $token_1$, fee information, and the value of k as its arguments. It emits a `PoolCreated()` event.

The deployer of a pool can determine its swap-fee structure. Each pool has two fees; f_{S0} and f_{S1} which define the swap fee, in tenths of basis points, when swapping from $token_1$ to $token_0$ and from $token_0$ to $token_1$, respectively.

Additionally, the ZeroSlipFactory exposes read-only functions for getting the number of pools through `allPoolsLength()` and `allPools()` to get the address of the pool at the given index.

The deployer of the ZeroSlipFactory can also set a new fee recipient, i.e. the recipient of the LP withdrawal fee, through `setFeeRecipient()`. The fee recipient can withdraw fees with the `withdrawFees()` function.

4.2 ZeroSlipPool

The ZeroSlipPool implements a new trading pair, or *pool*, for a pair of tokens with the given parameters when called from the ZeroSlipFactory. The ZeroSlipPool is the logic contract for a given pool. There may exist multiple pools for two tokens in order to, among other reasons, accommodate different fee structures and values of k. ZeroSlipPool is also a ZeroSlipERC20 and implements the interfaces of ZeroSlipERC20 as described in 4.3.

ZeroSlipPool exposes various functions for reading the state of and interacting with the contract. The most notable externally/publicly callable functions are:

`mint(_stakeholder, _tokenIn, _amountIn)` transfers *_tokenIn* from the *_stakeholder* to the pool in the amount defined in *_amountIn* and sends an appropriate amount of LPT to the *_stakeholder*.

`firstMint(_stakeholder, _tokenIn, _amountIn)` similar to the `mint()` function but only callable once. This function should be used when the first LPT are minted to avoid issues with re-entrancy attacks. Not using the function for the first mint may cause issues with fairness of LP reward distribution later.

`burn(_stakeholder, _amountIn, _tokenOut)` reduces the LPT supply by burning *_amountIn* tokens from the *_stakeholder* and sends the *_stakeholder* an appropriate amount of the token defined in *_tokenOut*. *_tokenOut* must be one of the tokens in the pool and there must be a sufficient balance of those tokens in the pool to honour the claim. Thus, it may be necessary to make calls to both sides of the pool to withdraw if the LP provides large amounts of liquidity in the pool. Burning LPT is subject to fees, as defined in the beginning of section 4.

`swap(_stakeholder, _tokenIn, _amountIn)` exchanges *_amountIn* of *_tokenIn* to the other asset supported by the pool. Subject to fees, as defined in the beginning of section 4.

It should be noted that the ZeroSlipPool implements most of its functions with as few checks as possible in order to minimise gas-expenditures. Consequently, calling the functions with wrong parameters may lead to irrecoverable losses. Routers, or contracts routing orders to the zsAMM, will be implemented to provide safer means for interacting with the zsAMM.

The uncommon names of the inner functions are due to gas optimisations stemming from indexing order reordering and decreased non-zero data.

4.3 ZeroSlipERC20

ZeroSlipERC20 is the ERC20 contract used for the representation of LPT. It is a standard OpenZeppelin ERC20 with 18 decimals and modifications to only its name and symbol.

5. Use cases

ZeroSlippage's zsAMM is primarily designed for computer-driven use. Such use includes, but is not limited to: a) serving as a liquidity source for arbitrage bots to enable efficient shifting from one equal asset to another; b) serving as a liquidity source for aggregators; c) serving as an interchange for the implementation of particular types of bridge designs or other protocols that require the exchange of one asset for another.

The zsAMM is provided as a public good where pools implement a zero-fee structure and liquidity providers do not withdraw their funds. In such pools, users do not incur any fees but LPs invariably lose the deposited funds should they burn the LPT.

Additionally, zsAMM may be used by human actors. However, it is recommended to use the less gas-efficient routers intended for this purpose when interacting with the protocol manually.

The zsAMM may also be used for various other uses, such as:

<u>1. Migrating from one token to another with the same properties</u>. Assuming $token_1$ is the new token and $token_0$ is the old token, a pool can be set up and $token_1$ supplied in the amount of the token supply. Setting $k = 1$, $f_{S0} = 100\,000$, and $f_{S1} = 0$ allows token holders to exchange old tokens in a 1:1 ratio to the new token, but not vice versa due to the 100,000 tenths of a basis point (i.e., 100%) swap fee rate. Burning all the LPT minted at pool creation by sending them to the 0-address ensures the deployer does not profit from the migration. Swapping the token to the new one is free (net of gas costs) to the old token holders.

<u>2. Migrating from one token to another with modified properties</u>. Similar to no. 1, should the issuer of a token want to, for example, change the denomination of their asset, they could create a pool where $k \neq 1$.

<u>3. Migration incetivisation</u>. Lowering the users' barrier of migrating from one asset to another; e.g., in the case of fiat-backed stablecoins, a stablecoin issuer could set the fee to exchange a competing stablecoin to their stablecoin to zero while setting the reverse fee high to discourage the reverse activity. The stablecoin issuer could then redeem the competitor's tokens and mint more of their own token in order to increase the circulating supply of their token.

<u>4. CLOB replication</u>. The zsAMM can be used to replicate a CLOB or Uniswap v3 style AMMs by deploying multiple zsAMMs for the same asset pair at different quote prices. However, unlike in CLOBs usually, the makers will pay a fee for removing their quote through the withdrawal fee mechanism. The zsAMM `mint()` and `burn()` functions are gas-competitive with the swapping costs in modern DEXes *(see Appendix A)*, making posting and removing quotes economically feasible.

5. OTC trading. Following from the CLOB-like properties, the zsAMM can be used as an OTC trading tool.

6. Peg protection. A reference asset, such as a rebasing liquid-staking derivative *("rLSD")*, real-world asset, or a stablecoin can improve its peg to the underlying asset by creating pairs with the reference asset itself or similar assets. For example, if an rLSD provider wants its token to be readily exchangeable and priced equal to Ether, it can set up a zsAMM pool with low swap fees and $k = 1$ to support the price stability.

7. Binary options. A zsAMM can be set up to replicate a binary option. For example, to create a binary call option, its writer can create a zsAMM where k is set to the strike price of the asset. They can then move the LPT to a time-lock contract that returns the LPT to the writer at expiry. Option buyers can redeem the binary option at any point in time by trading in a special ERC20 for the asset against which the binary option was written. Furthermore, the issue of the ERC20 used for redeeming the option can be efficiently sold & marketed *a priori* using a zsAMM.

6. Future versions & modifications

In the zsAMM implementation described in this paper, k is defined at contract creation and cannot be changed after the initialisation. Future versions of zsAMM could include a *dynamic zero price impact* AMM, which reads the value k from an external source, such as an oracle. However, this application would not strictly be a zero slippage AMM as it would be possible for a trade to be executed at a different price than the price when the order was sent in.

This downside could be mitigated by, for example, executing trades only at given prices, effectively turning the AMM into a *dynamic zero slippage* AMM, or a *dzsAMM*. The dzsAMM could be useful, for example, for creating efficient trading pairs between various liquid staking derivatives. Such implementation would have very different risk parameters and assumptions when compared to the zsAMM.

Additionally, as referenced in this paper, we intend to provide routers for accessing the zsAMM through, for example, more common interfaces and through functions that implement important safety checks to improve the user experience of human users and developers. However, routers will always add a level of gas inefficiency and advanced users should consider interacting directly with the contract. Alternatively, various off-chain checks can be implemented via APIs to avoid the use of routers while guaranteeing a level of safety for the end-user.

7. Disclaimer

As of the publication date of this paper the zsAMM core contracts have not been released for production and are subject to changes. Prior to entering production, the zsAMM will be subjected to rigorous evaluation and the functions or mechanisms introduced in this paper may change materially. When interacting with any ZeroSlip contracts, make sure you understand the differences between the theoretical and practical implementation.

A yellow paper delving into the final technical implementation may be produced later.

This paper is for general information purposes only. It does not constitute investment advice or recommendation or solicitation to buy or sell any investment or asset and should not be used in the evaluation of the merits of making an investment decision.

This paper does not provide legal, accounting, or financial advice and any losses incurred through the direct, indirect, consequential, or special use of the document are expressly disclaimed.

This paper reflects the current opinions of the author and is not on behalf of Ape Capital, LLC.

8. Acknowledgements

A huge thank you to everyone who provided feedback on the whitepaper. Especially, thank you to my degen frens and all the hedgefund bros who provided important and insightful feedback. Thanks to the anons and pseudonymous apes for comprehensive comments and for innovating the use of the zsAMM in CLOB-like applications and as a binary options system. Thanks to the various contributors for feedback on mechanism design and insights into aggregators, bots, and trading systems.

9. References

Adams, H., Zinsmeister, N., & Robinson, D. (2020, March). *Uniswap v2 Core*. whitepaper.pdf. Retrieved January, 2023, from https://uniswap.org/whitepaper.pdf

Adams, H., Zinsmeister, N., Salem, M., Keefer, R., & Robinson, D. (2021, March). *Uniswap v3 Core*. Uniswap v3 Core. Retrieved January, 2023, from https://uniswap.org/whitepaper-v3.pdf

Curve.fi. (2021, June). *Curve Documentation* (1.0.0). Curve Documentation. Retrieved January, 2023, from https://curve.readthedocs.io/_/downloads/en/latest/pdf/

Mota, M. (2021, July 17). *Understanding StableSwap (Curve)*. Understanding StableSwap (Curve). Retrieved January, 2023, from https://miguelmota.com/blog/understanding-stableswap-curve/

Appendix A: Gas Cost Overview

The zsAMM prioritises computational efficiency in its swap functions. The swap functions forfeit various security checks to improve gas efficiency at the cost of potential losses to the trader if they attempt to swap assets using wrong parameter values. For users who are unwilling to take these risks we recommend using routers.

Tables A1 and A2, present a comparison between the preliminary EVM transaction gas costs between common zsAMM operations and swaps in other common AMM protocols, respectively. Please note that the figures provided here may not be up-to-date with the deployed contracts and are provided for illustrative purposes only.

	`swap()`	`burn()`	`mint()`	`createZeroSlipPool()`
Gas*	77,658	85,521	88,912	3,252,862

Table A1: Preliminary EVM gas costs comparison between common zsAMM operations

** These are preliminary results from early testing. Actual results may differ.*

Protocol			**Gas savings using zsAMM function**		
Name	**Swap function***	**Gas****	`swap()`	`burn()`	`mint()`
Curve Crypto V2	exchange	212,279 (323,252)	63.4% (76.0%)	59.7% (73.5%)	58.1% (72.5%)
Curve StableSwap	exchange	121,359 (238,757)	36.0% (67.5%)	29.5% (64.2%)	26.7% (62.8%)
Uniswap v2	Router01.swapExactTokensForTokens	105,654 (127,530)	26.5% (39.1%)	19.1% (32.9%)	15.8% (30.3%)
	Router02.swapExactTokensForTokens	98,980 (135,543)	21.5% (42.7%)	13.6% (36.9%)	10.2% (34.4%)
Uniswap v3	SwapRouter.ExactInputSingle	110,186 (148,309)	29.5% (47.6%)	22.4% (42.3%)	19.3% (40.0%)
	NonfungiblePositionManager.Mint	325,526 (486,608)	76.1% (84.0%)	73.7% (82.4%)	72.7% (81.7%)

Table A2: Preliminary zsAMM EVM gas costs comparison against common AMM protocols

** Transactions involving including two tokens only*

*** Empirical results, lowest (highest) result shown*

Appendix B: Symbols and Formulas

F_S	Swap fee rate. Defined in tenths of a basis point (1/100 000, i.e. 0.0010%).
$F_{Sa} = \{F_S = 0 ? 0 : n F_S + 1\}$	Absolute swap fee. Where n is the amount of $token_{in}$. Defined in units of $token_{in}$.
F_{S0}	Swap fee rate when exchanging $token_1$ to $token_0$. Defined in tenths of a basis point.
F_{S1}	Swap fee rate when exchanging $token_0$ to $token_1$. Defined in tenths of a basis point.
$F_W = \{F_S = 0 ? 1 : \frac{F_S}{4}\}$	Withdrawal fee rate. The amount charged from LPs for exiting the pool. Defined in tenths of a basis point. F_S is the swap fee rate applied to $token_{out}$.
$F_{Wa} = n F_W + 1$	Absolute withdrawal fee. Where n is the amount of $token_{out}$. Defined in units of $token_{out}$.
k	The exchange rate between two assets. Given as a multiple of 10^6 within the smart contracts. Must be greater than or equal to 1.
$LPT = \frac{RV}{Count\ of\ LPT}$	LPT value formula. Quoted in $token_0$.
$RV = reserve_0 + k \times reserve_1$	Reserve value. Quoted in $token_0$.
$token_0$	One of the two assets accepted by a zsAMM pool. The default quote asset.
$token_1$	One of the two assets accepted by a zsAMM pool.
$token_0 = k \times token_1$	The zsAMM invariant. All swaps must act in accordance with the invariant, net of fees.
$token_{in}$	The token that is provided by a trader or an LP to the pool.
$token_{out}$	The token that is withdrawn by a trader or an LP from the pool.
$reserve_0$	Reserve balance of $token_0$; the total amount of $token_0$ in the pool.
$reserve_1$	Reserve balance of $token_1$; the total amount of $token_1$ in the pool.