#include "hook\_api.h"

// Constants for Drippy Token and tax configuration

#define DRIPPY\_CURRENCY "DRIPPY" // Currency code for Drippy Token

#define TAX\_RATE 500 // 5% (in basis points, 10000 = 100%)

#define ISSUER\_ACCOUNT "rDrippyIssuerTest" // Issuer account (Testnet placeholder)

#define TEAM\_WALLET "rTeamWallet" // Team wallet (Testnet placeholder)

#define AMM\_POOL "rAMMPool" // AMM pool (Testnet placeholder)

#define STATE\_KEY\_TAX\_CONFIG "tax\_config"

// Tax configuration structure

typedef struct {

uint16\_t buy\_nft\_share; // 30% = 3000 basis points

uint16\_t buy\_token\_share; // 50% = 5000

uint16\_t buy\_team\_share; // 20% = 2000

uint16\_t sell\_lp\_token; // 40% = 4000

uint16\_t sell\_lp\_xrp; // 40% = 4000

uint16\_t sell\_team\_share; // 20% = 2000

} TaxConfig;

// Function to update tax configuration in Hook state

int64\_t update\_tax\_config(uint32\_t reserved) {

TaxConfig config = {3000, 5000, 2000, 4000, 4000, 2000}; // Set tax splits

hook\_api::state\_set(&config, sizeof(TaxConfig), (uint8\_t\*)STATE\_KEY\_TAX\_CONFIG, 32);

return hook\_api::accept(reserved);

}

// Main Hook function

int64\_t hook(uint32\_t reserved) {

// Step 1: Initialize - Verify transaction type (Payment only)

uint8\_t txn\_type[2];

hook\_api::get\_field(txn\_type, 2, sfTransactionType);

if (txn\_type[0] != 0 || txn\_type[1] != 0) { // Payment = 0

return hook\_api::accept(reserved); // Ignore non-Payment transactions

}

// Get amount and currency details

uint8\_t amount[48];

int64\_t amount\_len = hook\_api::get\_field(amount, 48, sfAmount);

uint8\_t currency[20];

hook\_api::get\_currency(currency, 20, amount);

uint8\_t issuer[20];

hook\_api::get\_issuer(issuer, 20, amount);

// Step 1: Check if transaction involves Drippy Token

if (!hook\_api::buffer\_equal(currency, (uint8\_t\*)DRIPPY\_CURRENCY, 20) ||

!hook\_api::buffer\_equal(issuer, (uint8\_t\*)ISSUER\_ACCOUNT, 20)) {

return hook\_api::accept(reserved); // Ignore non-Drippy transactions

}

// Step 2: Check exclusion list

uint8\_t sender[20], recipient[20];

hook\_api::get\_account(sender, 20, sfAccount);

hook\_api::get\_account(recipient, 20, sfDestination);

uint8\_t excluded = 0; // Simplified: No exclusion list for Testnet

if (excluded) {

return hook\_api::accept(reserved); // Skip tax for excluded accounts

}

// Step 3: Determine transaction type (Buy or Sell)

int is\_buy = 0; // Simplified logic for Testnet

if (hook\_api::buffer\_equal(recipient, (uint8\_t\*)AMM\_POOL, 20)) {

is\_buy = 0; // Sell: User to AMM

} else if (hook\_api::buffer\_equal(sender, (uint8\_t\*)AMM\_POOL, 20)) {

is\_buy = 1; // Buy: AMM to user

} else {

return hook\_api::accept(reserved); // Ignore if neither buy nor sell

}

// Step 4: Apply tax

int64\_t drops = hook\_api::get\_drops(amount); // Get amount in drops

int64\_t tax\_drops = (drops \* TAX\_RATE) / 10000; // 5% tax

int64\_t net\_drops = drops - tax\_drops; // Amount after tax

// Store tax amount in Hook state for distribution

uint8\_t tax\_key[32] = "tax\_amount";

hook\_api::state\_set(&tax\_drops, 8, tax\_key, 32);

// Step 5–6: Distribute tax (Buy or Sell)

return distribute\_tax(reserved, is\_buy, tax\_drops);

}

// Function to distribute tax based on transaction type

int64\_t distribute\_tax(uint32\_t reserved, int is\_buy, int64\_t tax\_drops) {

if (is\_buy) {

// Buy tax distribution (updated as per bridging solution)

int64\_t lp\_xrp = (tax\_drops \* 10) / 100; // 0.5% XRP to LP

int64\_t lp\_drippy = (tax\_drops \* 10) / 100; // 0.5% DRIPPY to LP

int64\_t team = (tax\_drops \* 20) / 100; // 1% to team

int64\_t nft\_holders = (tax\_drops \* 20) / 100; // 1% to NFT holders

int64\_t token\_holders = (tax\_drops \* 40) / 100; // 2% to token holders

// Distribute to LP (mock swap for Testnet)

int64\_t xrp\_acquired = lp\_xrp; // Placeholder: No swap on Testnet

uint8\_t lp\_payment1[48], lp\_payment2[48];

hook\_api::prepare\_payment(lp\_payment1, (uint8\_t\*)AMM\_POOL, xrp\_acquired); // XRP

hook\_api::prepare\_payment(lp\_payment2, (uint8\_t\*)AMM\_POOL, lp\_drippy, (uint8\_t\*)DRIPPY\_CURRENCY, (uint8\_t\*)ISSUER\_ACCOUNT); // DRIPPY

hook\_api::emit(lp\_payment1, 48);

hook\_api::emit(lp\_payment2, 48);

// Distribute to team wallet

uint8\_t team\_payment[48];

hook\_api::prepare\_payment(team\_payment, (uint8\_t\*)TEAM\_WALLET, team);

hook\_api::emit(team\_payment, 48);

// Distribute to NFT holders (mock for Testnet, assumes 1 holder with 5+ NFTs)

uint8\_t nft\_holder[20] = "rNFTHolderTest"; // Placeholder

uint8\_t nft\_payment[48];

hook\_api::prepare\_payment(nft\_payment, nft\_holder, nft\_holders); // Single holder for Testnet

hook\_api::emit(nft\_payment, 48);

// Distribute to token holders (mock for Testnet, assumes 1 holder)

uint8\_t token\_holder[20] = "rTokenHolderTest"; // Placeholder

int64\_t holder\_share = token\_holders; // Single holder for Testnet

uint8\_t token\_payment[48];

hook\_api::prepare\_payment(token\_payment, token\_holder, holder\_share);

hook\_api::emit(token\_payment, 48);

} else {

// Sell tax distribution (updated as per bridging solution)

int64\_t lp\_xrp = (tax\_drops \* 20) / 100; // 1% XRP to LP

int64\_t lp\_drippy = (tax\_drops \* 20) / 100; // 1% DRIPPY to LP

int64\_t team = (tax\_drops \* 15) / 100; // 0.75% to team

int64\_t nft\_holders = (tax\_drops \* 15) / 100; // 0.75% to NFT holders

int64\_t token\_holders = (tax\_drops \* 30) / 100; // 1.5% to token holders

// Distribute to LP (mock swap for Testnet)

int64\_t xrp\_acquired = lp\_xrp; // Placeholder: No swap on Testnet

uint8\_t lp\_payment1[48], lp\_payment2[48];

hook\_api::prepare\_payment(lp\_payment1, (uint8\_t\*)AMM\_POOL, xrp\_acquired); // XRP

hook\_api::prepare\_payment(lp\_payment2, (uint8\_t\*)AMM\_POOL, lp\_drippy, (uint8\_t\*)DRIPPY\_CURRENCY, (uint8\_t\*)ISSUER\_ACCOUNT); // DRIPPY

hook\_api::emit(lp\_payment1, 48);

hook\_api::emit(lp\_payment2, 48);

// Distribute to team wallet

uint8\_t team\_payment[48];

hook\_api::prepare\_payment(team\_payment, (uint8\_t\*)TEAM\_WALLET, team);

hook\_api::emit(team\_payment, 48);

// Distribute to NFT holders (mock for Testnet, assumes 1 holder)

uint8\_t nft\_holder[20] = "rNFTHolderTest"; // Placeholder

uint8\_t nft\_payment[48];

hook\_api::prepare\_payment(nft\_payment, nft\_holder, nft\_holders); // Single holder for Testnet

hook\_api::emit(nft\_payment, 48);

// Distribute to token holders (mock for Testnet, assumes 1 holder)

uint8\_t token\_holder[20] = "rTokenHolderTest"; // Placeholder

int64\_t holder\_share = token\_holders; // Single holder for Testnet

uint8\_t token\_payment[48];

hook\_api::prepare\_payment(token\_payment, token\_holder, holder\_share);

hook\_api::emit(token\_payment, 48);

}

// Step 7: Update state (log transaction for debugging)

uint8\_t log\_key[32] = "last\_tax\_distributed";

hook\_api::state\_set(&tax\_drops, 8, log\_key, 32);

// Step 8: Finalize

return hook\_api::accept(reserved); // Accept if all payments succeed

}