# Synthesizer: Repository Structure

This document provides a detailed overview of the [Synthesizer](synthesizer-terminology.md#synthesizer) codebase file structure and organization.

## Directory Tree

packages/frontend/synthesizer/src/  
├── evm.ts # Extended EVM class  
├── interpreter.ts # Dual execution engine  
├── constructors.ts # EVM factory functions  
├── message.ts # Transaction message wrapper  
├── opcodes/  
│ ├── functions.ts # EVM opcode handlers  
│ └── synthesizer/  
│ └── handlers.ts # Synthesizer opcode handlers  
├── adapters/  
│ └── synthesizerAdapter.ts # External API interface  
├── tokamak/  
│ ├── core/  
│ │ ├── synthesizer/  
│ │ │ └── index.ts # Main Synthesizer class  
│ │ ├── handlers/  
│ │ │ ├── stateManager.ts # State management  
│ │ │ ├── operationHandler.ts # Arithmetic/logic ops  
│ │ │ ├── dataLoader.ts # External data (storage, env, etc.)  
│ │ │ ├── memoryManager.ts # Memory aliasing resolution  
│ │ │ └── bufferManager.ts # LOAD/RETURN buffer management  
│ │ └── finalizer/  
│ │ ├── index.ts # Finalizer orchestrator  
│ │ ├── permutation.ts # Wire map generation  
│ │ └── placementRefactor.ts # Wire size optimization  
│ ├── pointers/  
│ │ ├── stackPt.ts # Symbolic stack  
│ │ ├── memoryPt.ts # 2D memory tracker  
│ │ └── dataPointFactory.ts # Symbol factory  
│ ├── types/ # TypeScript type definitions  
│ ├── constant/ # Constants & subcircuit mappings  
│ └── utils/ # Utility functions

## Core Components

### EVM Extension Layer

#### evm.ts

* **Purpose**: Extended EthereumJS EVM class
* **Key Addition**: Synthesizer instance integration
* **Role**: Coordinates dual execution (EVM + Synthesizer)

#### interpreter.ts

* **Purpose**: Bytecode execution engine
* **Key Addition**: Dual state management (Stack/[StackPt](synthesizer-terminology.md#stackpt), Memory/[MemoryPt](synthesizer-terminology.md#memorypt))
* **Role**: Orchestrates opcode-by-opcode execution and consistency checks

#### constructors.ts

* **Purpose**: Factory functions for creating EVM instances
* **Key Addition**: RPC-based state manager integration
* **Role**: Entry point for Synthesizer execution

#### message.ts

* **Purpose**: Transaction message wrapper
* **Role**: Encapsulates transaction data for EVM execution

### Opcode Handlers

#### opcodes/functions.ts

* **Purpose**: Original EthereumJS opcode handlers
* **Role**: Standard EVM execution logic (value processing)

#### opcodes/synthesizer/handlers.ts

* **Purpose**: Tokamak Synthesizer opcode handlers
* **Role**: Circuit generation logic ([symbol processing](synthesizer-terminology.md#symbol-processing))

### Synthesizer Core

#### tokamak/core/synthesizer/index.ts

* **Purpose**: Main Synthesizer class (Facade pattern)
* **Key Methods**:
  + placeArith() - Arithmetic operations
  + loadAuxin() - Auxiliary inputs
  + loadStorage() - Storage access
  + loadEnvInf() - Environment info
  + loadBlkInf() - Block info
  + place() - Generic placement creation
* **Role**: Unified interface for all Synthesizer operations

### Handler Classes

#### tokamak/core/handlers/stateManager.ts

* **Purpose**: Circuit state management
* **Key Data**:
  + placements - Map of all [subcircuit](synthesizer-terminology.md#subcircuit) instances
  + placementIndex - Sequential ID counter
  + subcircuitInfoByName - Subcircuit metadata
* **Role**: Central registry for circuit construction

#### tokamak/core/handlers/operationHandler.ts

* **Purpose**: Arithmetic and logic operation handling
* **Key Methods**:
  + placeArith() - Map operations to subcircuits (ALU1, ALU2, etc.)
  + createOutput() - Generate output symbols
* **Role**: Translates EVM operations to circuit [placements](synthesizer-terminology.md#placement)

#### tokamak/core/handlers/dataLoader.ts

* **Purpose**: External data loading (storage, environment, block info)
* **Key Methods**:
  + loadStorage() - Storage data with caching
  + loadEnvInf() - Environment information
  + loadBlkInf() - Block information
  + loadCalldata() - Transaction calldata
* **Role**: Manages LOAD buffer (Placement 0 and 2)

#### tokamak/core/handlers/memoryManager.ts

* **Purpose**: Memory operation and [aliasing](synthesizer-terminology.md#data-aliasing) resolution
* **Key Methods**:
  + placeMemoryToStack() - Load memory with aliasing
  + combineMemorySlices() - Reconstruct from fragments
  + applyShift() - Generate SHR/SHL subcircuits
  + applyMask() - Generate AND subcircuits
* **Role**: Handles complex memory data dependencies

#### tokamak/core/handlers/bufferManager.ts

* **Purpose**: LOAD/RETURN [buffer](synthesizer-terminology.md#buffer-placements) management
* **Key Methods**:
  + addWireToInBuffer() - Add to input buffers ([PUB\_IN](synthesizer-terminology.md#pub-in-and-pub-out), [PRV\_IN](synthesizer-terminology.md#prv-in-and-prv-out))
  + addWireToOutBuffer() - Add to output buffers ([PUB\_OUT](synthesizer-terminology.md#pub-in-and-pub-out), [PRV\_OUT](synthesizer-terminology.md#prv-in-and-prv-out))
* **Role**: Bridge between external values and internal symbols

### Finalizer Components

#### tokamak/core/finalizer/index.ts

* **Purpose**: Orchestrates output file generation
* **Process**:
  1. Optimize placements (PlacementRefactor)
  2. Generate wire connections (Permutation)
  3. Calculate witness (outputPlacementVariables)
  4. Write output files
* **Role**: Converts circuit state to backend-compatible format

#### tokamak/core/finalizer/permutation.ts

* **Purpose**: [Wire](synthesizer-terminology.md#wire) map and [witness](synthesizer-terminology.md#witness) generation
* **Key Methods**:
  + buildPermutation() - Create wire connection groups
  + outputPermutation() - Generate permutation.json
  + outputPlacementVariables() - Generate placementVariables.json
  + generateSubcircuitWitness() - Calculate witness using WASM
* **Role**: Produces three critical output files

#### tokamak/core/finalizer/placementRefactor.ts

* **Purpose**: Circuit optimization
* **Key Optimizations**:
  + Wire size adjustments
  + Redundant placement removal
  + Wire index normalization
* **Role**: Prepares circuit for efficient proving

### Data Structures

#### tokamak/pointers/stackPt.ts

* **Purpose**: Symbolic stack implementation
* **Data Structure**: Array of [DataPt](synthesizer-terminology.md#datapt-data-point) symbols
* **Role**: Tracks stack operations symbolically

#### tokamak/pointers/memoryPt.ts

* **Purpose**: 2D memory tracking (offset × time)
* **Data Structure**: Map<timestamp, {memOffset, containerSize, dataPt}>
* **Key Method**: getDataAlias() - Identifies overlapping writes
* **Role**: Enables memory [aliasing](synthesizer-terminology.md#data-aliasing) resolution

#### tokamak/pointers/dataPointFactory.ts

* **Purpose**: DataPt symbol creation
* **Key Method**: create() - Generate unique symbols
* **Role**: Factory pattern for symbol instantiation

### Type Definitions

#### tokamak/types/

* **Purpose**: TypeScript type definitions
* **Key Types**:
  + DataPt - Symbolic data point
  + PlacementEntry - Subcircuit instance
  + Placements - Map of all placements
  + DataAliasInfo - Memory aliasing metadata
  + SubcircuitNames - Subcircuit type names
  + ArithmeticOperator - Operation names

### Constants and Mappings

#### tokamak/constant/

* **Purpose**: System-wide constants
* **Key Constants**:
  + SUBCIRCUIT\_MAPPING - Operation → Subcircuit mapping
  + INITIAL\_PLACEMENT\_INDEX - Starting ID (4)
  + BUFFER\_PLACEMENT\_IDS - 0 (PUB\_IN), 1 (PUB\_OUT), 2 (PRV\_IN), 3 (PRV\_OUT)

### Utilities

#### tokamak/utils/

* **Purpose**: Helper functions
* **Examples**:
  + BigInt conversions
  + Byte manipulation
  + Array utilities
  + Validation helpers

### External Interface

#### adapters/synthesizerAdapter.ts

* **Purpose**: Public API for external consumers
* **Key Function**: execSynthesizer(txHash, rpcUrl, outputPath)
* **Role**: User-facing interface for transaction processing