# Synthesizer: Key Terminology

This document defines key terms and concepts specific to the Tokamak Synthesizer system.

## Table of Contents

* [Auxiliary Input (Auxin)](#auxiliary-input-auxin)
* [Block Info (BlkInf)](#block-info-blkinf)
* [Buffer Placements](#buffer-placements)
* [Circom](#circom)
* [Data Aliasing](#data-aliasing)
* [DataPt (Data Point)](#datapt-data-point)
* [Environment Info (EnvInf)](#environment-info-envinf)
* [Finalizer](#finalizer)
* [Instance](#instance)
* [Limb](#limb)
* [MemoryPt](#memorypt)
* [Permutation](#permutation)
* [Placement](#placement)
* [PRV\_IN and PRV\_OUT](#prvin-and-prvout)
* [PUB\_IN and PUB\_OUT](#pubin-and-pubout)
* [QAP Compiler](#qap-compiler)
* [R1CS](#r1cs)
* [Selector](#selector)
* [StackPt](#stackpt)
* [Subcircuit](#subcircuit)
* [Symbol Processing](#symbol-processing)
* [Synthesizer](#synthesizer)
* [Wire](#wire)
* [Wire Index](#wire-index)
* [Witness](#witness)

## Auxiliary Input (Auxin)

Hardcoded constants from the bytecode that are loaded directly as DataPts without going through buffer placements. Used primarily for PUSH operations where the value is embedded in the contract bytecode.

**Example**:

PUSH1 0x20 // Loads 0x20 as auxiliary input

**Related**: [DataPt](#datapt-data-point)

## Buffer Placements

Four special placements (IDs 0-3) that serve as interfaces between the external world and the circuit’s internal symbol system. All external data must enter through input buffers and all outputs must exit through output buffers.

**The four buffers**:

* **0**: PUB\_IN (Public Input Buffer)
* **1**: PUB\_OUT (Public Output Buffer)
* **2**: PRV\_IN (Private Input Buffer)
* **3**: PRV\_OUT (Private Output Buffer)

**Related**: [PUB\_IN and PUB\_OUT](#pubin-and-pubout), [PRV\_IN and PRV\_OUT](#prvin-and-prvout)

## Data Aliasing

The problem of overlapping memory regions where multiple writes affect the same memory location. MemoryPt tracks all overlapping writes and generates circuits to correctly reconstruct memory values by combining multiple DataPts with appropriate shifts and masks.

**Example**:

MSTORE(0x00, valueA) // Writes to 0x00-0x20  
MSTORE(0x10, valueB) // Overlaps at 0x10-0x20  
MLOAD(0x00) // Must reconstruct: valueA[0x00-0x10] | valueB[0x10-0x20]

**Related**: [MemoryPt](#memorypt)

## DataPt (Data Point)

A symbolic representation of data in the circuit. Each DataPt contains:

* value: The actual numeric value (for consistency checking)
* source: Placement ID where this data originated
* wireIndex: Unique identifier within the source placement
* sourceSize: Size of the data in bytes

DataPts enable traceability of all data transformations throughout the circuit.

**Example**:

{  
 value: 0x20n,  
 source: 5, // From placement 5  
 wireIndex: 2, // Third wire in that placement  
 sourceSize: 32  
}

**Related**: [Wire](#wire), [Symbol Processing](#symbol-processing)

## Environment Info (EnvInf)

Transaction environment data that is loaded through the PUB\_IN buffer. Includes values like:

* msg.sender (ADDRESS)
* msg.value (CALLVALUE)
* tx.origin (ORIGIN)
* Calldata (CALLDATALOAD, CALLDATACOPY)

**Related**: [PUB\_IN and PUB\_OUT](#pubin-and-pubout)

## Finalizer

The component responsible for converting the Synthesizer’s internal state (placements and DataPts) into the three output files required for proof generation. The Finalizer:

1. Generates permutation.json by analyzing wire connections
2. Generates instance.json by extracting buffer values and creating witness arrays
3. Generates placementVariables.json by calculating witnesses for each placement using WASM

**Related**: [Output Files](./synthesizer-output-files.md)

## Instance

The public and private input/output values for the circuit, stored in instance.json. The instance contains:

* Buffer contents (publicInputBuffer, publicOutputBuffer, privateInputBuffer, privateOutputBuffer)
* Flattened witness arrays (a\_pub, a\_prv)

The instance is used by the Prover to generate the proof and by the Verifier to verify it.

**Related**: [Witness](#witness), [instance.json](./synthesizer-output-files.md#instancejson)

## Limb

A 128-bit portion of a 256-bit value. Since Circom uses a 254-bit finite field, 256-bit Ethereum values are split into two 128-bit limbs (lower and upper) to ensure all operations stay within the field size.

**Example**:

256-bit value: 0x123456789ABCDEF0123456789ABCDEF0123456789ABCDEF0123456789ABCDEF0  
Lower limb: 0x123456789ABCDEF0123456789ABCDEF0  
Upper limb: 0x123456789ABCDEF0123456789ABCDEF0

**Related**: [Circom](#circom)

## MemoryPt

The symbolic equivalent of EVM memory that tracks all memory writes with timestamps. MemoryPt solves the data aliasing problem by maintaining a complete history of memory operations, allowing reconstruction of memory state at any point through circuit operations.

**Key features**:

* Timestamp-based tracking of all writes
* Data aliasing resolution for overlapping memory regions
* Lazy circuit generation (circuits created on MLOAD, not MSTORE)

**Related**: [Data Aliasing](#data-aliasing)

## Permutation

A mechanism to enforce wire equality constraints across placements. Wires that carry the same value are grouped into N-entry cycles in permutation.json. This is based on the PLONK permutation argument and ensures that connected wires have identical values.

**Example** (3-entry cycle):

[  
 {"row": 13, "col": 1, "X": 14, "Y": 3},  
 {"row": 14, "col": 3, "X": 15, "Y": 2},  
 {"row": 15, "col": 2, "X": 13, "Y": 1}  
]

**Interpretation**: Wire 13 in Placement 1 → Wire 14 in Placement 3 → Wire 15 in Placement 2 → back to Wire 13 in Placement 1 (cycle complete).

**Related**: [Wire](#wire), [Output Files](./synthesizer-output-files.md)

## Placement

A specific instance of a subcircuit with concrete input and output DataPts. Each placement represents one operation in the circuit and is assigned a unique sequential ID starting from 4 (IDs 0-3 are reserved for buffer placements). Placements form a directed acyclic graph (DAG) that represents the complete computation.

**Example**:

{  
 id: 5,  
 name: 'ALU1',  
 subcircuitId: 4,  
 usage: 'ADD',  
 selector: 0x02,  
 inPts: [selectorPt, input1Pt, input2Pt],  
 outPts: [resultPt]  
}

**Related**: [Subcircuit](#subcircuit), [DataPt](#datapt-data-point)

## PRV\_IN and PRV\_OUT

**PRV\_IN (Placement 2)**: Converts external private values into circuit symbols. Used for sensitive inputs like storage values and account state that should remain hidden.

**PRV\_OUT (Placement 3)**: Converts circuit symbols back to external private values. Used for private outputs like storage updates and internal state changes.

**Usage**: Only the Prover has access to these values; they are never revealed to the Verifier.

**Related**: [Buffer Placements](#buffer-placements)

## PUB\_IN and PUB\_OUT

**PUB\_IN (Placement 0)**: Converts external public values into circuit symbols. Used for publicly known inputs like calldata, block.number, msg.sender, and Keccak hash outputs.

**PUB\_OUT (Placement 1)**: Converts circuit symbols back to external public values. Used for publicly revealed outputs like return data, event logs, and Keccak hash inputs.

**Usage**: Both Prover and Verifier have access to these values.

**Related**: [Buffer Placements](#buffer-placements)

## QAP Compiler

The component that compiles Circom subcircuits into Quadratic Arithmetic Programs (QAPs) and provides pre-defined circuits to the Synthesizer. The QAP Compiler produces:

* WASM modules for witness calculation
* Constraint information for each subcircuit
* Setup parameters for the proof system
* Pre-compiled subcircuit library (ALU1, ALU2, ALU3, etc.) ready for use by the Synthesizer

**Repository**: [packages/frontend/qap-compiler](https://github.com/tokamak-network/Tokamak-zk-EVM/tree/main/packages/frontend/qap-compiler)

**Related**: [Circom](#circom), [Subcircuit](#subcircuit)

## R1CS

Rank-1 Constraint System - the mathematical representation of a circuit as a set of constraints of the form (A · witness) \* (B · witness) = (C · witness). Every Circom circuit is compiled into R1CS format, which is then used by the proof system.

**Key concept**: Wire 0 is always constant 1 in R1CS to enable expressing constant terms in constraints.

**Related**: [Circom](#circom), [Witness](#witness)

## Selector

A value used to choose which operation within a multi-operation subcircuit should be executed. Selectors are typically powers of 2 (e.g., 1 << 1n for ADD, 1 << 2n for MUL) to enable efficient bitwise selection in the circuit.

**Example** (ALU1 selectors):

* ADD: 0x02 (1 << 1)
* MUL: 0x04 (1 << 2)
* SUB: 0x08 (1 << 3)
* EQ: 0x100000 (1 << 20)
* ISZERO: 0x200000 (1 << 21)

**Related**: [Subcircuit](#subcircuit), [Subcircuit Mapping Table](./synthesizer-opcodes.md#appendix-subcircuit-mapping-table)

## StackPt

The symbolic equivalent of the EVM stack. Instead of storing concrete values, StackPt stores DataPts that represent symbolic references to circuit wires. Every stack operation (push, pop, dup, swap) operates on DataPts while the parallel EVM stack operates on values.

**Consistency check**: After each opcode, the Synthesizer verifies that stack.values == stackPt.values to ensure correctness.

**Related**: [DataPt](#datapt-data-point), [Symbol Processing](#symbol-processing)

## Subcircuit

A reusable circuit template defined in Circom that implements a specific operation or set of operations. Subcircuits are compiled into WASM for witness generation.

**Related**: [Placement](#placement), [Circom](#circom)

## Symbol Processing

The fundamental approach of the Synthesizer where all data is treated as symbolic references (DataPts) rather than concrete values. This allows tracking the complete provenance and transformation of every piece of data through the circuit, enabling zero-knowledge proof generation.

**Contrast with EVM**:

* **EVM**: result = a + b (values only)
* **Synthesizer**: resultPt = placement(ADD, [aPt, bPt]) (symbols with provenance)

**Related**: [DataPt](#datapt-data-point)

## Synthesizer

The core component that converts Ethereum transaction execution into zk-SNARK circuit representations. The Synthesizer runs in parallel with the standard EVM execution, tracking all operations as symbolic relationships rather than concrete values. It generates three output files: permutation.json, instance.json, and placementVariables.json.

**Repository**: [packages/frontend/synthesizer](https://github.com/tokamak-network/Tokamak-zk-EVM/tree/main/packages/frontend/synthesizer)

**Related**: [Synthesizer Concepts](./synthesizer-concepts.md), [Execution Flow](./synthesizer-execution-flow.md), [Official Documentation](https://tokamak.notion.site/Synthesizer-documentation-164d96a400a3808db0f0f636e20fca24)

## Wire

A connection point in a circuit that carries a value. Each wire in a placement is identified by its wire index. Wires connect placements together, forming the circuit graph. The permutation.json file describes how wires are connected across placements.

**Related**: [Wire Index](#wire-index), [Permutation](#permutation)

## Wire Index

A unique identifier for a wire within a specific placement. Wire indices start from 0 (constant 1 in Circom convention) and increment for each input, output, and internal signal in the subcircuit.

**Standard ordering** (Circom convention):

1. Wire 0: Constant 1
2. Wires 1-N: Output signals
3. Wires N+1-M: Input signals
4. Wires M+1+: Internal signals

**Related**: [Wire](#wire), [DataPt](#datapt-data-point)

## Witness

The complete set of values for all wires in a circuit that satisfies all constraints. The witness includes inputs, outputs, and all intermediate computation values. For each placement, the witness is calculated using the subcircuit’s WASM module.

**Components**:

* **Public witness** (a\_pub): Values from PUB\_IN and PUB\_OUT buffers
* **Private witness** (a\_prv): Values from PRV\_IN and PRV\_OUT buffers
* **Placement witness**: All wire values for each subcircuit instance

**Related**: [Instance](#instance), [placementVariables.json](./synthesizer-output-files.md#placementvariablesjson)