Understanding Arrays in SystemVerilog

Arrays are fundamental data structures in SystemVerilog. They enable efficient storage and manipulation of homogeneous data. This presentation will cover various array types and their applications.

Essential Data Structures

Store collections of homogeneous data efficiently.

Enhanced Functionality

Dynamic sizing, associative indexing, and memory organization.

Broad Applications

Used in RTL modeling and verification environments.

Introduction to Arrays in SystemVerilog

Homogeneous Data Storage

Arrays are essential for storing collections of similar data types.

SystemVerilog Enhancements

Adds dynamic sizing, associative indexing, and flexible memory organization.

RTL and Verification

Widely applied in both hardware description and verification tasks.

Types of Arrays in SystemVerilog

1

By Size

- Static Arrays
- Dynamic Arrays

2

By Layout

- Packed Arrays
- Unpacked Arrays

3

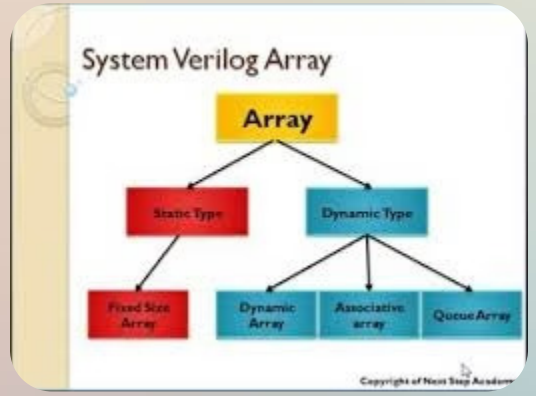
By Access Pattern

- Associative Arrays
- Queues

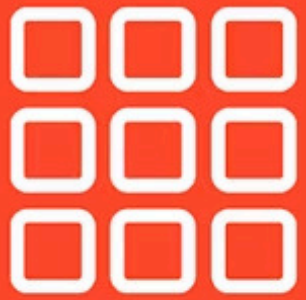
4

Hybrid Arrays

Combinations of packed and unpacked dimensions.



Block Storage



Data stored in fixed-size blocks

Static Arrays

Fixed Size

Their size is determined and fixed at compile time.

Known Elements

Used when the exact number of elements is known in advance.

Explicit Dimensions

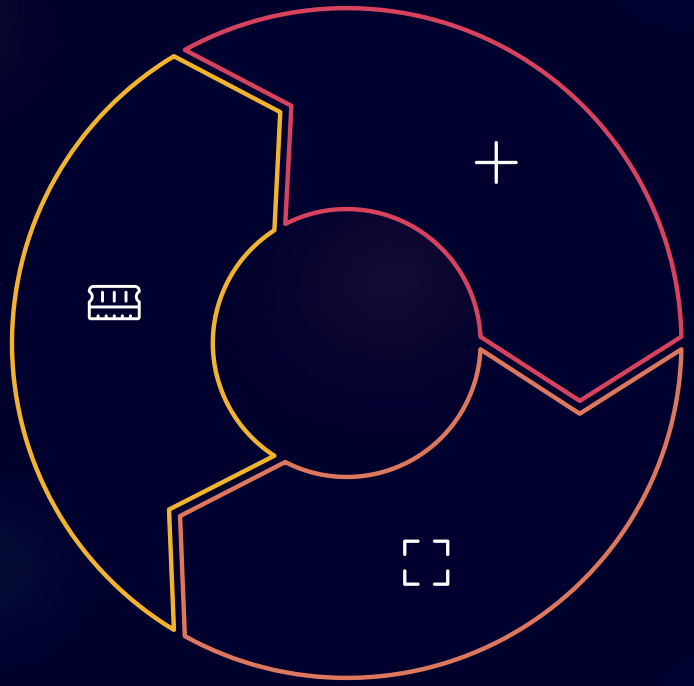
Declared with predefined dimensions in the code.

```
int static_array[10]; // 10-element integer array
```

Dynamic Arrays

Runtime Sizing

Size can be changed during simulation execution.



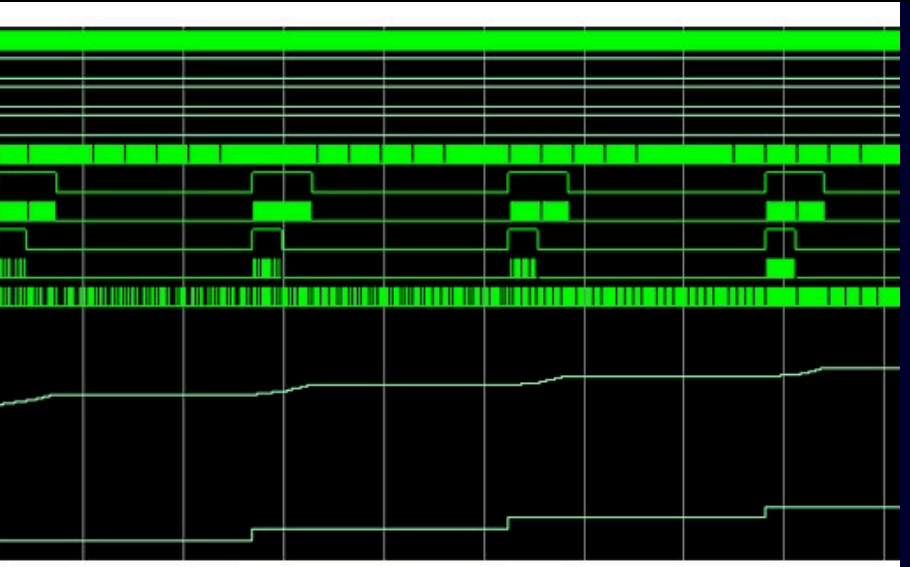
Allocate Memory

Use **new[]** operator for allocation.

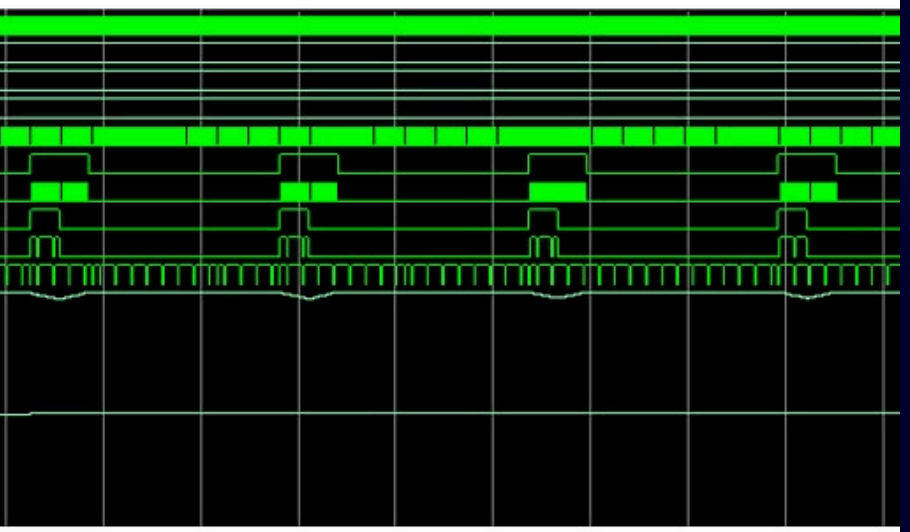
Resizable Data

Ideal for data that frequently changes size.

```
int dynamic_array[];
dynamic_array = new[5]; // Allocate 5 elements
dynamic_array = new[10](dynamic_array); // Resize and preserve
```



(a)



(b)

Packed Arrays

Packed arrays store bits contiguously in memory. They are fundamental for representing hardware structures like vectors and buses.

1

Contiguous Bits

Elements are stored as a single block of bits.

2

Hardware Modeling

Perfect for modeling data buses, vectors, and registers.

3

Dimension First

Declared with dimensions before the variable name.

```
logic [7:0] byte_data;
logic [3:0][7:0] packed_bytes; // 4 packed bytes
```

Unpacked Arrays

Separate Memory

Each element is allocated as a distinct memory unit.

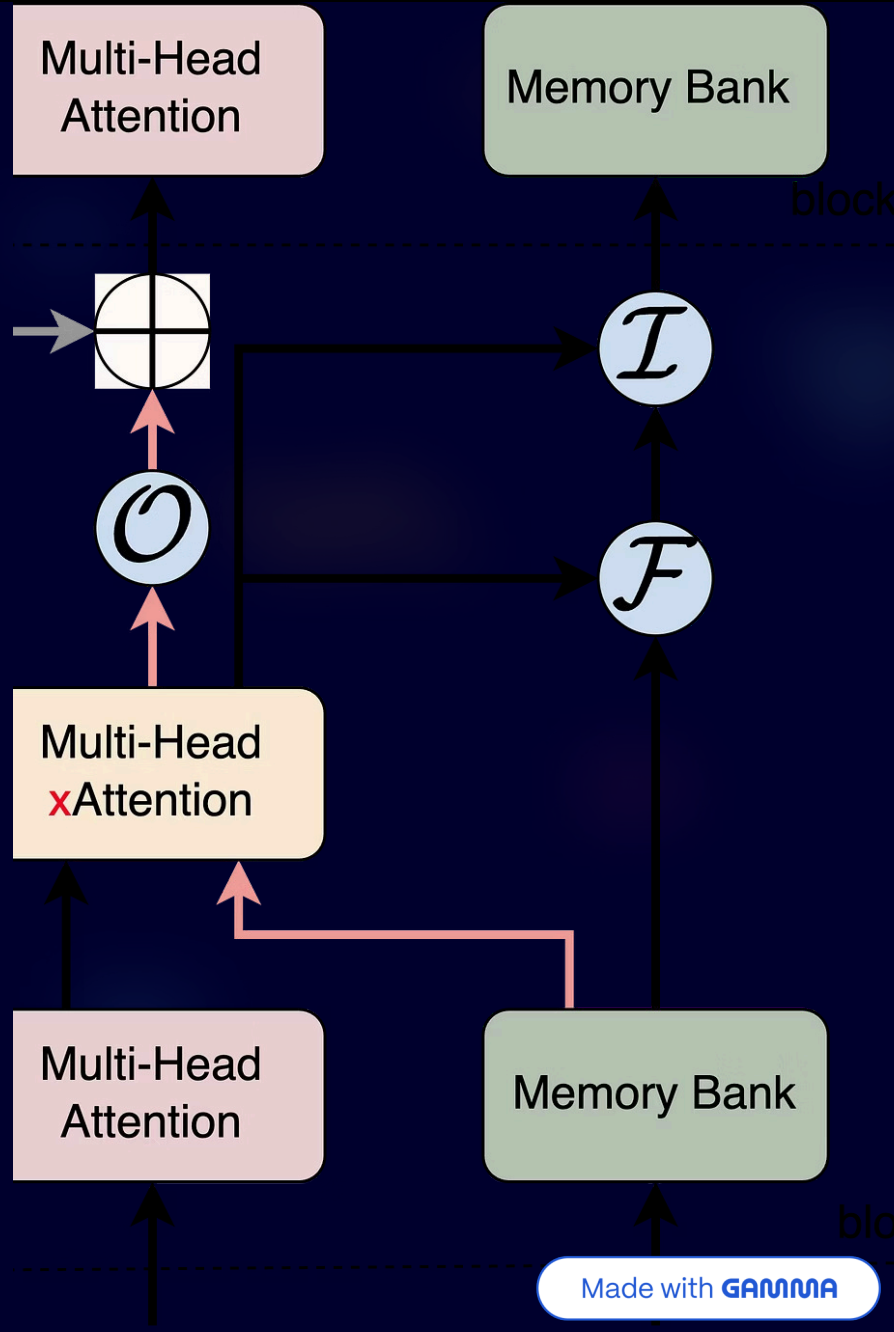
Dimension After

Dimensions are specified after the variable name.

Flexible Types

Can store any data type, including complex structures.

```
int array[5]; // Integer array
byte matrix[2][3]; // 2D matrix of bytes
```



Hybrid Arrays

Hybrid arrays combine both packed and unpacked dimensions, allowing for efficient representation of complex data structures.

Combine Dimensions

Integrate both packed and unpacked array declarations.

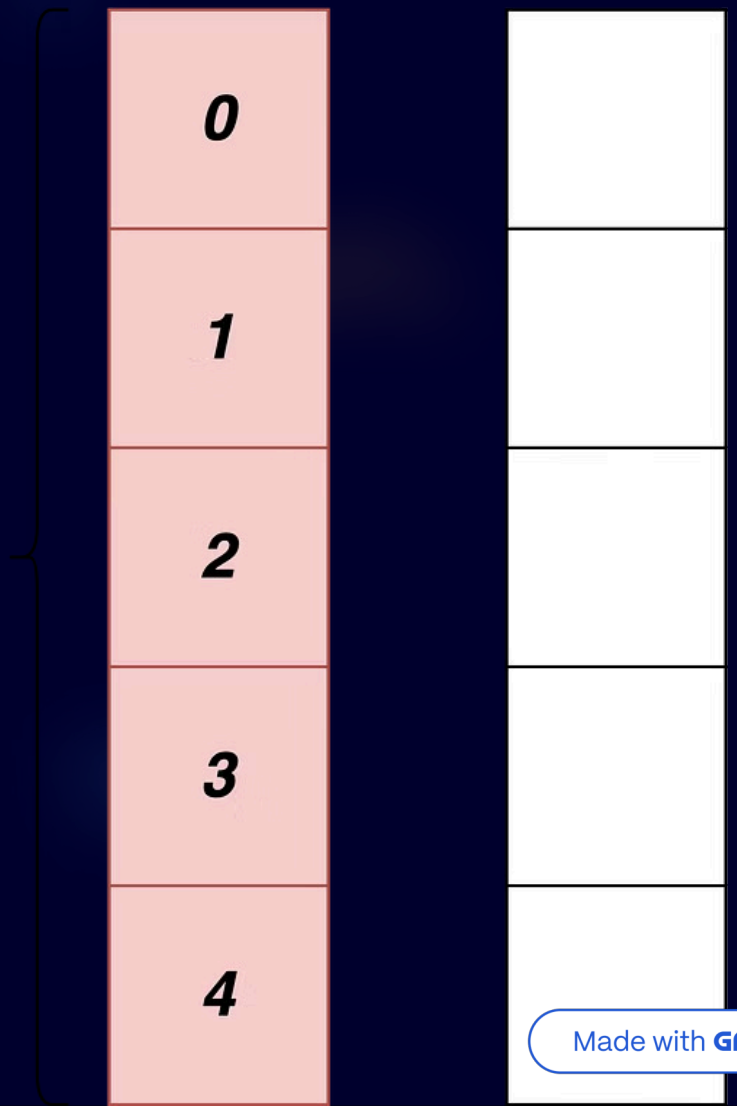
Structured Data

Ideal for representing structured binary data types.

Flexible Design

Enables versatile and efficient memory organization.

```
logic [7:0] data_bus[4]; // 4 unpacked elements, each 8-bit packed
```



Associative Arrays

Key-Value Access

Access elements using keys instead of numerical indices.

Sparse Storage

Memory is only allocated for assigned elements, saving space.

Flexible Keys

Keys can be of any data type, providing high flexibility.

```
string employee_map[int]; // int key -> string value
employee_map[101] = "Alice";
```