SystemVerilog for Verification

BASIC DATA TYPES – PART III

Agenda

- ✓ Structure
- ✓ Union
- ✓ Arrays
- ✓ Dynamic Array
- ✓ Associative Array
- ✓ Queue

Structure (Collection of Data Types)

Structure Declaration :

```
struct {
  logic [3:0] addr;
  int data;
} IR;

IR.addr = 4'b0010;
```

User-defined structure :

```
typedef struct {
  logic [3:0] addr;
  int data;
} instruction;

instruction IR;
```

Packed structure :

```
struct packed {
  logic [3:0] addr;
  int data;
} IR;

IR = 36'hA013BCD34;
```

Signed Packed structure :

```
struct packed signed {
  bit [3:0] addr;
  int data;
} IR;  //signed 2-state

IR = 36'hA013BCD34;
```

Unsigned Packed structure :

```
struct packed unsigned {
  logic [3:0] addr;
  int data;
} IR;  //signed 4-state

IR = 36'hA013BCD34;
```

structure literal :

```
typedef struct {
  int a;
  shortreal b;
} ab;
ab c;
c = '{0,0.0}
```

Structure Exercise

- Declare a user-defined packed structure for below register table. Make a code that reads and writes the value through register with sensitivity of register address change. Provide various register addresses from testbench at different time. Improper access should shout error.

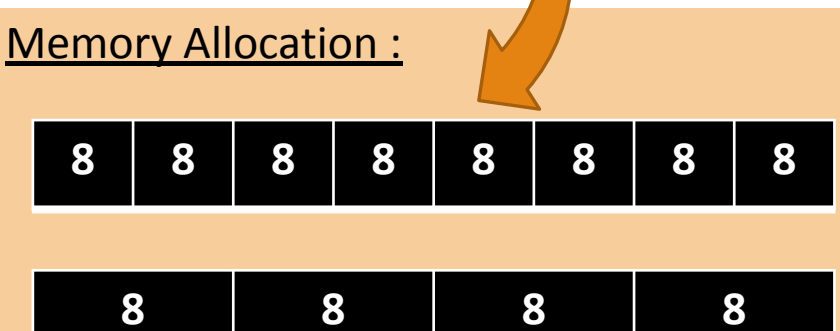
Address	Name	Access (R –Read, W-Write, RO – Read Only, WO – Write Only)
0xEEF1_0000	REG0	R/W
0xEEF1_0004	REG1	R/W
0xEEF1_0008	REG2	RO
0xEEF1_000C	REG3	R/W
0xEEF1_0010	REG4	WO

Union

- A single piece of storage that can be accessed using one of the named member data types.

User-defined Unpacked (Default) union :

```
typedef union {
  int a;
  shortreal b;
} uni;
uni u;
u.b = 0.0;
```



User-defined Packed Union :

All Datatype of integral type having same size

```
typedef packed union {
  bit [31:0] mem1;
  bit [3:0] [7:0] mem2;
} packuni;
packuni p;
```

Memory Allocation :

Tagged Union :

- Original Union Loophole – updated using value of one member datatype, read using another.
- Tagged Union – value can be read only by one having tagged of lastly updated.

```
typedef union tagged packed{int a1, int a2 } u;
u u1; u1 = tagged a1 5; int b = u1.a1;
```

Arrays

Packed Arrays :

- Treated as single vector
- byte, shortint, int, longint, integer, time

```
byte c2;
integer a1;
```

- **bit signed** [7:0] c2;//size before variable

```
logic signed [31:0] a1;
```

Unpacked Array :

- size after variable name
- **logic** b1 [7:0]
- **int** a1 [7:0]

// 2D array - packed int; unpacked byte

Memory :

```
logic [7:0] mem [0:255]
mem[5] = 0;
data = mem[addr]
```

Array Literals:

int a [3:0]; → a = '{4{3}} = '{3,3,3,3}

int b [0:1] [0:2] → b = '{'{1,2,3},'{3{4}}}

1	2	3
4	4	4

Dynamic Array

- Unpacked array, whose size can be set/change at runtime.
- declared by [] Ex. - int a [];

Method	Description

new[]()	Set/Change size of array, default value of array members = 0
	set → int a [] = new [4]; //a size 4 → int a [] = new [4] (b); //copy b to a
	change → int a [] = new [8] (a); //copy old a to new a, increase size to 8

size()	Returns current size of dynamic array → int j = addr.size(); → addr = new[addr.size() * 4] (addr); // quadruple addr array

delete()	Empties an array
	int ab = new [4];
	ab.delete(); // ab having size = 0

Dynamic Array Exercise

- Using dynamic array, allocate the array size of 15 location, assign random values to all. Now double the array size by keeping content of initial 15 locations same as previous. Use foreach loop to print content of each.

  Note: search for $random in SV LRM.

Associative Array

- Dynamic array - good with contiguous data collection, whose size changes dynamically
- But when size of collection is unknown / Data space is sparse –Use Associative Array
- Index can be of any data type, say string.

  data_type array_name [index_type]

  Ex. int abc [string];

  integer def [*]; //unspecified index, can be anything

- Associative arrays do not have any storage allocated until it is used.

Associative Array

Method	Description
num(), size()	Returns number of entries in associative array
delete()	Removes entry at specific index
exists()	Checks whether element exists at specific index
first()	Returns the value of first index
last()	Returns the value of last index
next()	Returns the smallest index whose value is greater than given index
prev()	Returns the largest index whose value is lesser than given index

Associative Array Exercise

Add 70 integer values at random locations(between 1 to 150) of an integer associative array.

1. Check whether value at index 10 and 56 exists.
2. Print the value at first index along with index.
3. Print the value at last index along with index.
4. Check the array size.
5. Delete 15^{th}, 30^{th} and 65^{th} index if they exists.
6. Print array size again.

Queue

- variable-size, ordered collection of homogeneous elements.

```
byte q1 [$] ;                    // queue of bytes, $ represents last.
string q2 [$] = {“hello”} ;   // queue of strings initialized to “hello”
int q3 [$] = {3,2,7} ;          // initialized queue of integers
```

Queue

Method	Description
size()	Returns number of items in queue
insert()	Inserts given item at specific index
delete()	Deletes item at specific index
pop_front()	Removes and returns the first element of queue
pop_back()	Removes and returns the last element of queue
push_front()	Inserts element at front of queue
push_back()	Inserts element at last of queue

Queue Exercise

- Make a FIFO (First In First Out) of 128x32 using queue & its methods

  Hint: methods – push_front, pop_back

- Make a LIFO (Last In First Out) of 256x64 using queue & its methods

  Hint: methods – push_front, push_back

Thank You