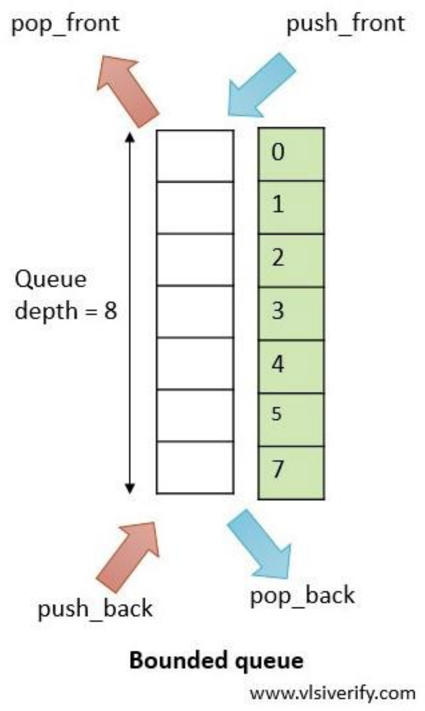
SystemVerilog Queues

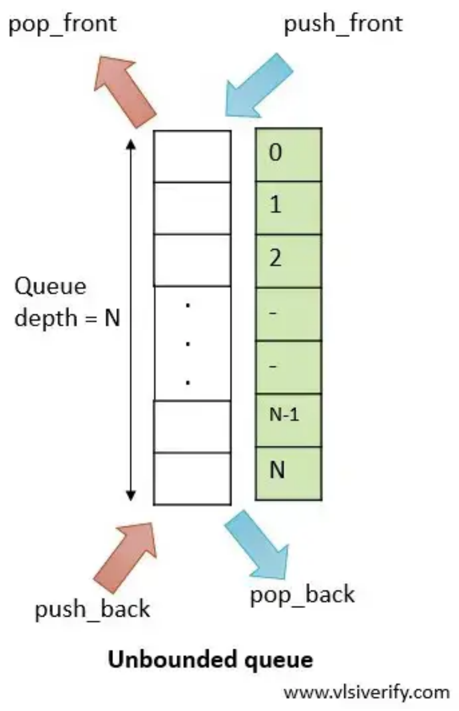
A queue is a variable size and ordered collection of elements (homogeneous element).

To understand it is considered the same as a single-dimensional unpacked array that grows and reduces automatically if it is a bounded queue.

Types of queues in SystemVerilog

1. Bounded queue: Queue having a specific size or a limited number of entries.
2. Unbounded queue: Queue having non-specific queue size or unlimited entries.





Declaration of a queue in SystemVerilog

```
data_type  <queue_name> [$];
```

For Example:

```
bit q_1[$];     // Unbounded queue of bit
byte q_2[$];    // Unbounded queue of byte
int q_3 [$:9];  // Bounded queue with qsize = 10

int q_4[$] = {5,6,7};
```

SystemVerilog Queue methods

Methods (functions)	**Description**
insert (<index>, <item>)	Inserts an item at a specified index.
1. delete(<index>) 2. delete	1. Deletes an item at a specified index 2. Deletes all elements in the queue.
size()	If the queue is not empty, return the number of items in the queue. Otherwise, it returns 0.
push_back(<item>)	Inserts an item at the end of the queue.
pop_back()	Returns and removes the last item of the queue.
push_front(<item>)	Inserts an item at the front of the queue.
pop_front()	Returns and removes the first item of the queue.

shuffle()	Shuffles items in the queue

SystemVerilog Queue Example

```
module queue_example;
  // declaration
  string animal_q[$];

  initial begin
    $display("Initial Size: animal_q = %0d", animal_q.size());

    animal_q = {"TIGER","LION"};
    $display("Size: animal_q = %0d", animal_q.size());
    $display("-----------------------");

    animal_q.insert(1, "ELEPHANT");
    animal_q.insert(3, "FOX");
    animal_q.insert(4, "ZEBRA");
    $display("Size: animal_q = %0d", animal_q.size());

    foreach(animal_q[i]) $display("animal_q[%0d] = %s", i, animal_q[i]);
    $display("-----------------------");

    $display("--- Access queue item ---");
    $display("The second element of animal_q = %s", animal_q[2]);
    $display("The fourth element of animal_q = %s", animal_q[4]);
    $display("-----------------------");

    $display("--- Delete queue item ---");
    animal_q.delete(2);
    foreach(animal_q[i]) $display("animal_q[%0d] = %s", i, animal_q[i]);
    $display("-----------------------");

    $display("--- Delete complete queue ---");
    animal_q.delete();
    $display("Size after queue deletion: animal_q size = %0d", animal_q.size());
    $display("-----------------------");


    animal_q = {"TIGER","LION"};

    $display("--- push_back methods ---");
    animal_q.push_back("ELEPHANT");
    foreach(animal_q[i]) $display("animal_q[%0d] = %s", i, animal_q[i]);
    $display("-----------------------");

    $display("--- push_front methods ---");
    animal_q.push_front("FOX");
    foreach(animal_q[i]) $display("animal_q[%0d] = %s", i, animal_q[i]);
    $display("-----------------------");

    $display("--- pop_back methods ---");
    animal_q.pop_back();
    foreach(animal_q[i]) $display("animal_q[%0d] = %s", i, animal_q[i]);
    $display("-----------------------");

    $display("--- pop_front methods ---");
    animal_q.pop_front();
    foreach(animal_q[i]) $display("animal_q[%0d] = %s", i, animal_q[i]);
    $display("-----------------------");
  end
endmodule
```

Output:

```
Initial Size: animal_q = 0
Size: animal_q = 2
-----------------------
Size: animal_q = 5
animal_q[0] = TIGER
animal_q[1] = ELEPHANT
animal_q[2] = LION
animal_q[3] = FOX
animal_q[4] = ZEBRA
-----------------------
--- Access queue item ---
The second element of animal_q = LION
The fourth element of animal_q = ZEBRA
-----------------------
--- Delete queue item ---
animal_q[0] = TIGER
animal_q[1] = ELEPHANT
animal_q[2] = FOX
animal_q[3] = ZEBRA
-----------------------
--- Delete complete queue ---
Size after queue deletion: animal_q size = 0
-----------------------
--- push_back methods ---
animal_q[0] = TIGER
animal_q[1] = LION
animal_q[2] = ELEPHANT
-----------------------
--- push_front methods ---
animal_q[0] = FOX
animal_q[1] = TIGER
animal_q[2] = LION
animal_q[3] = ELEPHANT
-----------------------
--- pop_back methods ---
animal_q[0] = FOX
animal_q[1] = TIGER
animal_q[2] = LION
-----------------------
--- pop_front methods ---
animal_q[0] = TIGER
animal_q[1] = LION
-----------------------
```

Example for shuffle method

Let’s see how the shuffle method shuffles queue’s items.

```
module queue_example;
  // declaration
  int num_q[$];

  initial begin
    for(int i = 0; i < 10; i++) num_q.push_back(i);
    $display("--- Before shuffle ---");
    foreach(num_q[i]) $display("num_q[%0d] = %0d", i, num_q[i]);
    num_q.shuffle();
    $display("----------------------");
    $display("--- After shuffle ---");
    foreach(num_q[i]) $display("num_q[%0d] = %0d", i, num_q[i]);
  end
endmodule
```

Output:

```
--- Before shuffle ---
num_q[0] = 0
num_q[1] = 1
num_q[2] = 2
num_q[3] = 3
num_q[4] = 4
num_q[5] = 5
num_q[6] = 6
num_q[7] = 7
num_q[8] = 8
num_q[9] = 9
----------------------
--- After shuffle ---
num_q[0] = 1
num_q[1] = 3
num_q[2] = 9
num_q[3] = 6
num_q[4] = 8
num_q[5] = 5
num_q[6] = 2
num_q[7] = 4
num_q[8] = 0
num_q[9] = 7
```

Array of queues

An array can store queues. In the below example,

array[0] stores a queue of even numbers.

array[1] stores a queue of odd numbers.

array[2] stores a queue of multiple hundreds.

Initialization of array of queues

- Based on array index

```
array[0] = {2, 4, 6, 8};
array[1] = {1, 3, 5, 7};
array[2] = {100, 200, 300};
```

- Without using an array index

```
array = '{ {2, 4, 6, 8},
           {1, 3, 5, 7},
           {100, 200, 300}
         };
```

Array of queues Example

```
module array_example;
  int array [3][$];

  initial begin
    //array[0] = {2, 4, 6, 8};
    //array[1] = {1, 3, 5, 7};
    //array[2] = {100, 200, 300};
    //or
    array = '{ {2, 4, 6, 8},
               {1, 3, 5, 7},
               {100, 200, 300}
             };

    // Print array of queues
    foreach (array[i,j]) $display("array[%0d][%0d] = %0d", i, j, array[i][j]);
    $display("------------------");

    array[0].push_back(10);
    array[1].push_back(9);
    array[2].push_back(400);

    $display("After push_back operation");
    // Print array of queues
    foreach (array[i,j]) $display("array[%0d][%0d] = %0d", i, j, array[i][j]);

  end
endmodule
```

Output:

```
array[0][0] = 2
array[0][1] = 4
array[0][2] = 6
array[0][3] = 8
array[1][0] = 1
array[1][1] = 3
array[1][2] = 5
array[1][3] = 7
array[2][0] = 100
array[2][1] = 200
array[2][2] = 300
------------------
After push_back operation
array[0][0] = 2
array[0][1] = 4
array[0][2] = 6
array[0][3] = 8
array[0][4] = 10
array[1][0] = 1
array[1][1] = 3
array[1][2] = 5
array[1][3] = 7
array[1][4] = 9
array[2][0] = 100
array[2][1] = 200
array[2][2] = 300
array[2][3] = 400
```

System Verilog Tutorials

- Data Types in SV
- SystemVerilog Arrays
- Dynamic Array in SV
- Associative array in SV
- Array manipulation methods
  - Array locator methods
  - Array ordering methods
  - Array reduction methods
  - Iterator index querying

PREVIOUS NEXT

Contact Us
Privacy Policy

Interview Questions

Verilog Interview Questions with Answers
SystemVerilog Interview Questions with Answers
UVM Interview Questions with Answers

Tutorials

Verilog Tutorials
SystemVerilog Tutorials
Functional Coverage Tutorials
Assertions Tutorials
UVM Tutorials
TLM Tutorials
RAL Tutorials

Follow Us

Learn More

ASIC Flows
Verilog Codes
Resources
Blogs

Support Us On