DAY-59
#100DAYSRTL

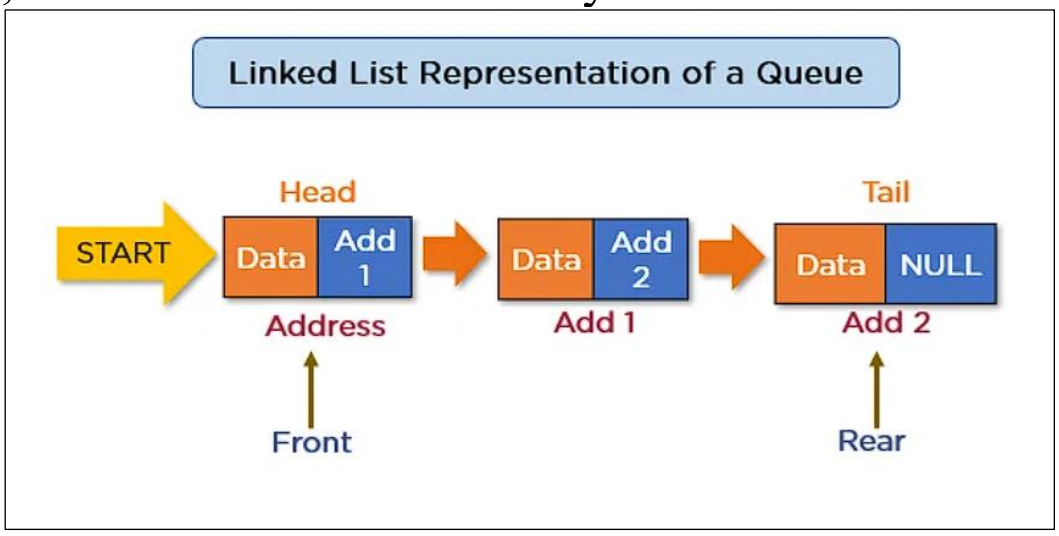
"System Verilog: Queues"

Introduction:-

The queue is a special data type in System Verilog that works on the principle of FIFO (First in First Out). Thus, the element that is stored first in the queue is retrieved first, just as the case in the real-life queue.

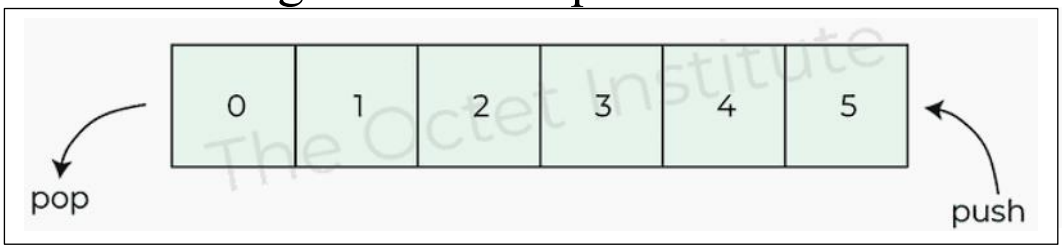
Queues:-

- Queues are based on the concept of a linked list in which the elements consist of 2 parts. 1st part is the value that needs to be stored, and the 2nd part is the address of the next element of the list. If the element is the last one, then the 2nd part is null otherwise it points to the next item.
- Queue in system Verilog, handles all the address-related operations internally which makes using queue a lot easier. Queue like an associative array has no specific size and the size grows and shrinks dynamically depending upon the number of elements stored. But in queues, the index is continuous in nature, unlike the associative arrays.



Concept of push and pop:-

- As discussed, queues work on the concept of FIFO, i.e., first in first out. This means that a new element can be added at the end of the queue. Also, the element which is first in queue means that it was inserted earlier than other elements and thus this element will be removed from the queue first.
- **Push** in queue means inserting a new element at the end of the queue.
- **Pop** in queue means getting the first element present in the queue and removing it from the queue.



Syntax:-

- `<element datatype> <queue_name> [$];`

Code Practising:-

```
module tb;
  bit [7:0] Odd[$];
  initial begin
    Odd[$+1]=8'd1;
    Odd[$+1]=8'd3;
    Odd[$+1]=8'd5;
    Odd[$+1]=8'd7;
    Odd[$+1]=8'd9;
    $display("Odd=%p",Odd);
    Odd='{8'd11,8'd13,8'd15};
    $display("Odd=%p",Odd);
  end
endmodule
```

Result:-

```
Odd='{1, 3, 5, 7, 9}
Odd='{11, 13, 15}
```

Important Note:-

Using the $symbol for adding new elements in the queue may not produce the same result in different simulators. Thus, it is recommended to use the built-in functions to add or remove elements from a queue so that the output is the same on all simulators.

Methods:-

1. **int size()** - returns the number of elements present in the queue.
2. **element_t pop_front()** - returns the first element present in the queue and deletes it.
3. **element_t pop_back()** - returns the last element present in the queue and deletes it.
4. **void push_front(element_t item)** - Inserts the item in the first position of the queue. Index of all other item shifts by 1.
5. **void push_back()** - inserts the item in the last position of the queue. All other items present in the queue are unaffected.
6. **void insert(int index, element_t item)** - inserts the item in the specified index.
7. **void delete([int index])** - if index is provided as input, it deletes the item present in specific index otherwise it delete the entire queue.

Code Practising:-

```
module queue_func_example();
    bit [7:0] queue [$]; // queue which can hold 8-bit data
    initial begin
        $display("Number of elements present in queue in %0d", queue.size());
        queue.push_back(34);
        queue.push_back(67);
        queue.push_back(123);
        $display("Values stored in queue after push_back are %p", queue);
        queue.push_front(212);
        $display("Values stored in queue after push_front are %p", queue);
        $display("Number of elements present in queue in %0d", queue.size());
        $display("Poped value from back = %0d", queue.pop_back());
        $display("Values stored in queue after pop_back are %p", queue);
        $display("Poped value from front = %0d", queue.pop_front());
        $display("Values stored in queue after pop_back are %p", queue);
        // Note that index starts from 0
        queue.insert(1, 56);
        $display("Values stored in queue after insert at 2nd position are %p", queue);
        queue.delete(1);   // deletes item at index 1
        $display("Values stored in queue after deleting item at 1st position are %p", queue);
        queue.delete();    // delets all the item
        $display("Values stored in queue after deleting all item are %p", queue);
    end
endmodule
```

Result:-

```
Number of elements present in queue in 0
Values stored in queue after push_back are '{34, 67, 123}
Values stored in queue after push_front are '{212, 34, 67, 123}
Number of elements present in queue in 4
Poped value from back = 123
Values stored in queue after pop_back are '{212, 34, 67}
Poped value from front = 212
Values stored in queue after pop_back are '{34, 67}
Values stored in queue after insert at 2nd position are '{34, 56, 67}
Values stored in queue after deleting item at 1st position are '{34, 67}
Values stored in queue after deleting all item are '{}
```

Example:-

- Let's suppose there is a process that samples data and sends it to the processing unit. It is not necessary that the data sent by the sampling unit would be processed immediately.
- In this scenario we don't know exactly how many elements we need to store thus dynamic array is not a good option. Also, an associative array is not good for this scenario as we don't need the elements after the element has been processed. In the case of an associative array, the size will increase, and it is slower.
- Thus, queues are the best option for this scenario as the sampling unit can push data once it has sampled it and the processing unit can pop it whenever it can process the data.

Code Practising:-

```
module queue_example2();
  bit [7:0] Tx,Rx;
  bit [7:0] buffer [$]; //FIFO
    // Transmitter Unit
  initial begin
  repeat (10) begin
        Tx= $urandom_range(255);
      $display("[%0t]Sampled value is %0d\n", $time,Tx);
      buffer.push_back(Tx);
        #10;
    end
    // Receiver unit
  repeat (10) begin
        $display("[%0t]Number of elements present in queue in %0d", $time, buffer.size());
        $display("[%0t]Values stored in buffer is %p", $time, buffer);
        Rx= buffer.pop_front();
        $display("[%0t]Recieved value is %0d\n", $time, Rx);
        #20;
    end
  end
endmodule
```

Result:-

```
[0]Sampled value is 205
:[5]Sampled value is 122
:[10]Sampled value is 62
:[15]Sampled value is 39
:[20]Sampled value is 230
:[25]Sampled value is 6
:[30]Sampled value is 54
:[35]Sampled value is 220
:[40]Sampled value is 175
:[45]Sampled value is 115
:[50]Number of elements present in queue in 10
[50]Values stored in buffer is '{205, 122, 62, 39, 230, 6, 54, 220, 175, 115}
[50]Recieved value is 205
[70]Number of elements present in queue in 9
[70]Values stored in buffer is '{122, 62, 39, 230, 6, 54, 220, 175, 115}
[70]Recieved value is 122
[90]Number of elements present in queue in 8
[90]Values stored in buffer is '{62, 39, 230, 6, 54, 220, 175, 115}
[90]Recieved value is 62
[110]Number of elements present in queue in 7
[110]Values stored in buffer is '{39, 230, 6, 54, 220, 175, 115}
[110]Recieved value is 39
[130]Number of elements present in queue in 6
[130]Values stored in buffer is '{230, 6, 54, 220, 175, 115}
[130]Recieved value is 230
[150]Number of elements present in queue in 5
[150]Values stored in buffer is '{6, 54, 220, 175, 115}
[150]Recieved value is 6
[170]Number of elements present in queue in 4
[170]Values stored in buffer is '{54, 220, 175, 115}
[170]Recieved value is 54
[190]Number of elements present in queue in 3
[190]Values stored in buffer is '{220, 175, 115}
[190]Recieved value is 220
[210]Number of elements present in queue in 2
[210]Values stored in buffer is '{175, 115}
[210]Recieved value is 175
[230]Number of elements present in queue in 1
[230]Values stored in buffer is '{115}
[230]Recieved value is 115
```