DAY-60
#100DAYSRTL

"System Verilog: Array Manipulations"

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 in the case of the real-life queue.

With Keyword:-

Basically, with keyword is used to pass a condition. This keyword is used with a function to put some constraint for the argument that we are passing as the function. These keywords are not only used with the array manipulation functions but also with some other constructs which we will see later in this series.

Syntax:-

```
function_name([argument]) with ([argument condition]);
```

Ordering Functions:-

1. **reverse()**:-This function reverses the order of the array. This means that the first element will be shifted to the last position and the last element will come to the first position.
2. **sort()**:-This function sorts the array in the ascending order. This can be used in conjunction with the keyword.
3. **rsort()**:-This function sorts the array in descending order. This also supports the keyword optionally.
4. **shuffle()**:-Randomly changes the order of the elements present in the array.

Note:- We cannot use the keyword with reverse and shuffle functions.

Code practicing:-

```
module ordering_example();
  int arr[] = '{40,20,70,30,40};
    int res[];
    initial begin
        arr.reverse();
        $display("arr.reverse() = %p", arr);
      arr.sort() with (item > 60);
        $display("arr.sort() + with = %p", arr);
        arr.sort();
        $display("arr.sort() = %p", arr);
        arr.rsort();
        $display("arr.rsort() = %p", arr);
        arr.shuffle();
        $display("arr.shuffle() = %p", arr);
    end
endmodule
```

Result:-

```
arr.reverse() = '{40, 30, 70, 20, 40}
arr.sort() + with = '{40, 20, 40, 30, 70}
arr.sort() = '{20, 30, 40, 40, 70}
arr.rsort() = '{70, 40, 40, 30, 20}
arr.shuffle() = '{20, 40, 70, 40, 30}
```

Searching Functions:-

find():-This function returns all the elements which satisfy a given condition.

find_index():-Returns the index of all the elements which satisfy a given condition. This is somewhat similar to find() but returns the index instead of actual elements.

find_first():-returns the first element which satisfies the given condition.

find_first_index():-returns the index of the first element which satisfies the given condition.

find_last():-returns the last element that satisfies the given condition.

find_last_index():-returns the index of the last element which satisfies the given condition.

Note:- The “with” clause is important with all these functions**.**

Code practicing:-

```
module find_example();
  int arr[] = '{8,4,6,2,7,4,9,3};
    int res[];
    initial begin
        res = arr.find(x) with (x > 5);
        $display("res = %p", res);
      res = arr.find() with (item > 6);
        $display("res = %p", res);
        res = arr.find_index(x) with (x > 5);
        $display("res = %p", res);
        res = arr.find_first(x) with (x > 5);
        $display("res = %p", res);
      res = arr.find_last(x) with (x > 5);
        $display("res = %p", res);
    end
endmodule
```

Result:-

```
res = '{8, 6, 7, 9}
res = '{8, 7, 9}
res = '{0, 2, 4, 6}
res = '{8}
res = '{9}
```

Some Other functions:-

1. **min()**:-Returns the smallest value of the array
2. **max()**:-Returns the largest value of the array
3. **unique()**:-Returns the array in which all the duplicate values are removed.
4. **unique_index()**:-Return the array of indices that contains a unique value.
5. **sum()**:-Returns the sum of all array elements
6. **product()**:-Returns the product of all array elements
7. **and()**:-Returns the bitwise AND (&) of all array elements
8. **or()**:-Returns the bitwise OR (|) of all array elements
9. **xor()**:-Returns the bitwise XOR (^) of all array elements

Code practicing:-

```
module reduction_example();
    bit[5:0] arr[] = '{12, 45, 23};
    int res;
    bit[5:0] red_res;
    int arr2[] = '{12, 45, 23, 89, 23, 43, 23, 78, 98};
    int res2[];
    initial begin
        res = arr.sum();
        $display("arr.sum() = %0d", res);
        res = arr.product();
        $display("arr.product() = %0d\n", res);
        red_res = arr.and();
        $displayb("arr = %p", arr);
        $display("arr.and() = %b", red_res);
        red_res = arr.or();
        $display("arr.or() = %b", red_res);
        red_res = arr.xor();
        $display("arr.xor() = %b", red_res);
        res2 = arr2.min();
        $display("arr.min() = %p", res);
        res2 = arr2.max();
        $display("arr.max() = %p", res);
        res2 = arr2.unique();
        $display("arr.unique() = %p", res);
        res2 = arr2.unique_index();
        $display("arr.unique_index() = %p", res);
    end
endmodule
```

Result:-

```
arr.sum() = 16
arr.product() = 4

arr = '{6'b001100, 6'b101101, 6'b010111}
arr.and() = 000100
arr.or() = 111111
arr.xor() = 110110
arr.min() = 4
arr.max() = 4
arr.unique() = 4
arr.unique_index() = 4
```