Shri B V V Sangha's

BASAVESHWAR ENGINEERING COLLEGE,BAGALKOTE

DEPARTMENT OF ELECTRONICS AND COMMUNICATION ENGINEERING

BVVS-BEC

Report on

"BUBBLE SORT USING VERILOG PROGRAMMING"

Associates:
1.Karthik P Jalagar USN:2BA20EC032 (Roll No:29)
2.Pratik B Joshi USN:2BA20EC054 (Roll No:49)

Div:A
Subject code: UEC543C

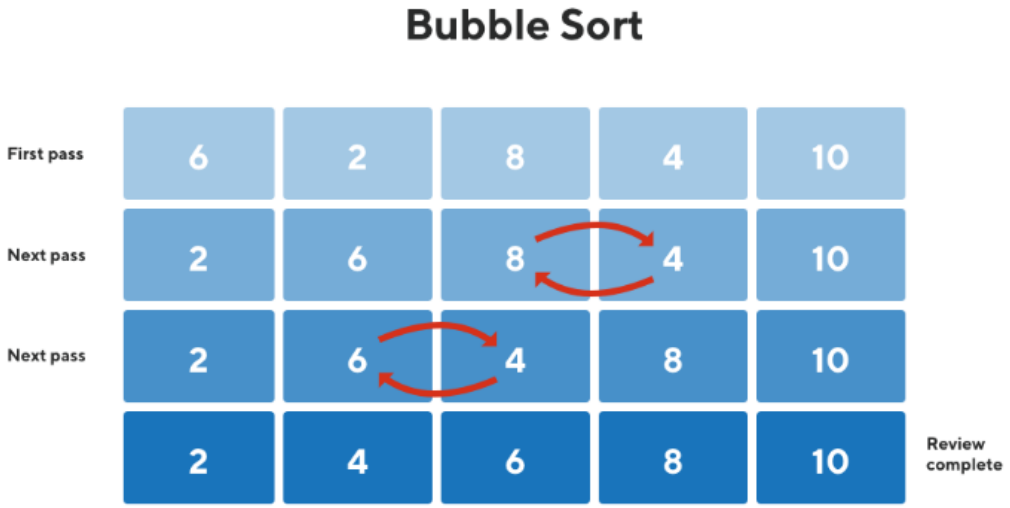
Submitted to: Prof.M.C.Aralimarad

Introduction:

What is Bubble sort?
=> Bubble sort is a basic algorithm for arranging a string of numbers or other elements in the correct order. The method works by examining each set of adjacent elements in the string, from left to right, switching their positions if they are out of order.The algorithm then repeats this process until it can run through the entire string and find no two elements that need to be swapped

What Does a Bubble Sort Look Like?
=> If a programmer or analyst wanted to arrange a series of numbers in ascending order, the bubble sort approach would look like the example pictured here.

The algorithm would review two items at a time, rearrange those not already in ascending order from left to right, and then continue to cycle through the entire sequence until it completed a pass without switching any numbers.It can be also used to sort in descending order also.



How does Bubble Sort Work?

Example=>

Input: arr[] = {5, 1, 4, 2, 8}

First Pass:

Bubble sort starts with very first two elements, comparing them to check which one is

greater.

(5 1 4 2 8) –> (1 5 4 2 8), Here, algorithm compares the first two elements, and swaps since 5 > 1.

(1 5 4 2 8) –> (1 4 5 2 8), Swap since 5 > 4

(1 4 5 2 8) –> (1 4 2 5 8), Swap since 5 > 2

(1 4 2 5 8) –> (1 4 2 5 8), Now, since these elements are already in order (8 > 5), algorithm does not swap them.

Second Pass:

Now, during second iteration it should look like this:

(1 4 2 5 8) –> (1 4 2 5 8)

(1 4 2 5 8) –> (1 2 4 5 8), Swap since 4 > 2

(1 2 4 5 8) –> (1 2 4 5 8)

(1 2 4 5 8) –> (1 2 4 5 8)

Third Pass:

Now, the array is already sorted, but our algorithm does not know if it is completed.

The algorithm needs one whole pass without any swap to know it is sorted.

(1 2 4 5 8) –> (1 2 4 5 8)

(1 2 4 5 8) –> (1 2 4 5 8)

(1 2 4 5 8) –> (1 2 4 5 8)

(1 2 4 5 8) –> (1 2 4 5 8)

Illustration:

i = 0	j	0	1	2	3	4	5	6	7
	0	5	3	1	9	8	2	4	7
	1	3	5	1	9	8	2	4	7
	2	3	1	5	9	8	2	4	7
	3	3	1	5	9	8	2	4	7
	4	3	1	5	8	9	2	4	7
	5	3	1	5	8	2	9	4	7
	6	3	1	5	8	2	4	9	7
i =1	0	3	1	5	8	2	4	7	9
	1	1	3	5	8	2	4	7	
	2	1	3	5	8	2	4	7	
	3	1	3	5	8	2	4	7	
	4	1	3	5	2	8	4	7	
	5	1	3	5	2	4	8	7	
i = 2	0	1	3	5	2	4	7	8	
	1	1	3	5	2	4	7		
	2	1	3	5	2	4	7		
	3	1	3	2	5	4	7		
	4	1	3	2	4	5	7		
i = 3	0	1	3	2	4	5	7		
	1	1	3	2	4	5			
	2	1	2	3	4	5			
	3	1	2	3	4	5			
i = 4	0	1	2	3	4	5			
	1	1	2	3	4				
	2	1	2	3	4				
i = 5	0	1	2	3	4				
	1	1	2	3					
i = 6	0	1	2	3					
		1	2						

Algorithm:

- **Run a nested for loop to traverse the input array using two variables i and j, such that $0 \leq i < n-1$ and $0 \leq j < n-i-1$**
- **If arr[j] is greater than arr[j+1] then swap these adjacent elements, else move on**
- **Print the sorted array.**

Bubble sort can be implemented to sort array either in ascending or descending order.

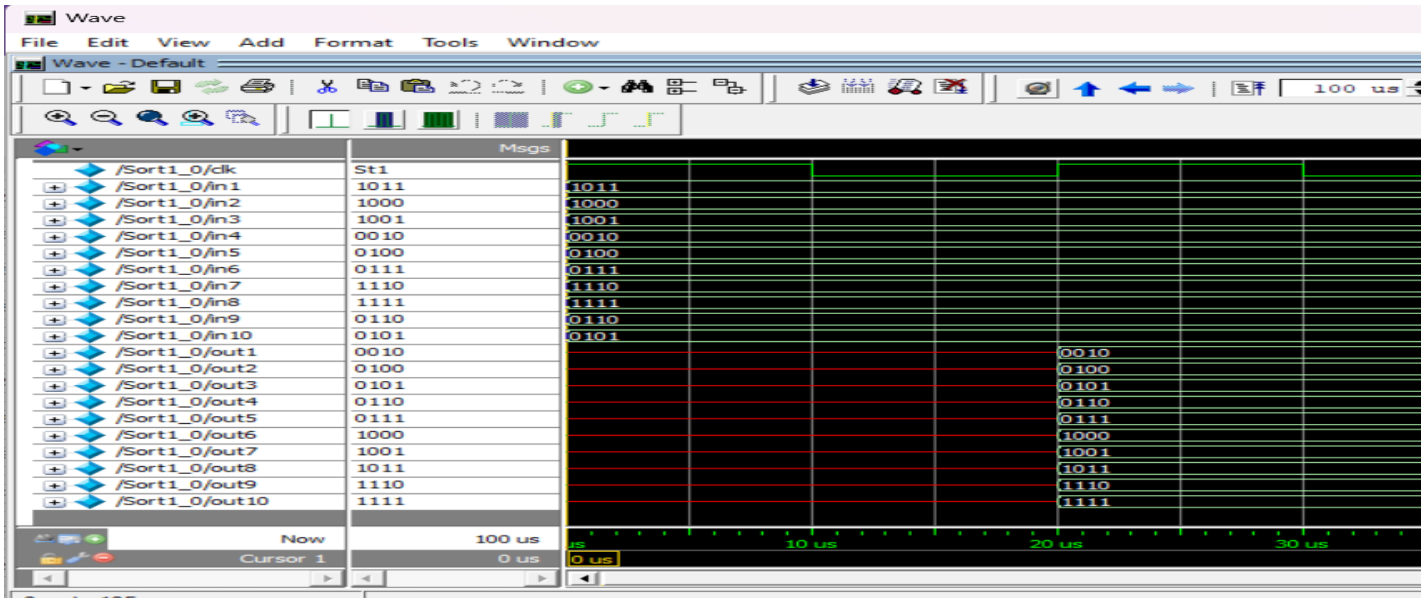
Verilog code to sort numbers in ascending order:

```
module Sort1_0(input wire clk,input wire [3:0]in1,in2,in3,in4,in5,in6,in7,in8,in9,in10,output reg [3:0]out1,out2,out3,out4,out5,out6,out7,out8,out9,out10);

reg [3:0]t0,t1,t2,t3,t4,t5,t6,t7,t8,t9,t10;

 always @(posedge clk)

 begin

 t1 <= in1;

t2 <= in2;

 t3 <= in3;

 t4 <= in4;

 t5 <= in5;

 t6 <= in6;

 t7 <= in7;

 t8<= in8;

 t9 <= in9;

 t10 <= in10;

end

integer i,j;

reg [3:0] temp;

reg [3:0] array [1:10];

always @*

begin

   array[1]=t1;
```

```
        array[2]=t2;
        array[3]=t3;
        array[4]=t4;
        array[5]=t5;
        array[6]=t6;
        array[7]=t7;
        array[8]=t8;
        array[9]=t9;
        array[10]=t10;
for (i=10; i>0; i=i-1)
begin
for (j=1; j<i; j=j+1)
begin
if (array[j] > array[j+1])
begin
temp = array[j];
array[j] = array[j+1];
array[j+1] = temp;
end
end
end
end
```

```
always @(posedge clk)
begin
out1 <= array[1];
out2 <= array[2];
out3 <= array[3];
out4 <= array[4];
out5 <= array[5];
out6 <= array[6];
out7 <= array[7];
out8 <= array[8];
out9 <= array[9];
out10 <= array[10];
end
endmodule
```

Output:



Verilog code to sort numbers in descending order:

```
module Sort1_0(input wire clk,input wire [3:0]in1,in2,in3,in4,in5,in6,in7,in8,in9,in10,output reg [3:0]out1,

out2,out3,out4,out5,out6,out7,out8,out9,out10);

 reg [3:0]t0,t1,t2,t3,t4,t5,t6,t7,t8,t9,t10;

 always @(posedge clk)

 begin

 t1 <= in1;

 t2 <= in2;

 t3 <= in3;

 t4 <= in4;

 t5 <= in5;

 t6 <= in6;

 t7 <= in7;

 t8<= in8;

 t9 <= in9;

 t10 <= in10;

end

integer i,j;

reg [3:0] temp;

reg [3:0] array [1:10];

always @*

begin
```

```
 array[1]=t1;
        array[2]=t2;
        array[3]=t3;
        array[4]=t4;
        array[5]=t5;
        array[6]=t6;
        array[7]=t7;
        array[8]=t8;
        array[9]=t9;
        array[10]=t10;
for (i=10; i>0; i=i-1)
begin
for (j=1; j<i; j=j+1)
begin
if (array[j] < array[j+1])
begin
temp = array[j];
array[j] = array[j+1];
array[j+1] = temp;
end
end
end
end
```

```
always @(posedge clk)
begin
out1 <= array[1];
out2 <= array[2];
out3 <= array[3];
out4 <= array[4];
out5 <= array[5];
out6 <= array[6];
out7 <= array[7];
out8 <= array[8];
out9 <= array[9];
out10 <= array[10];
end
endmodule
```

Output: