MINI PROJECT

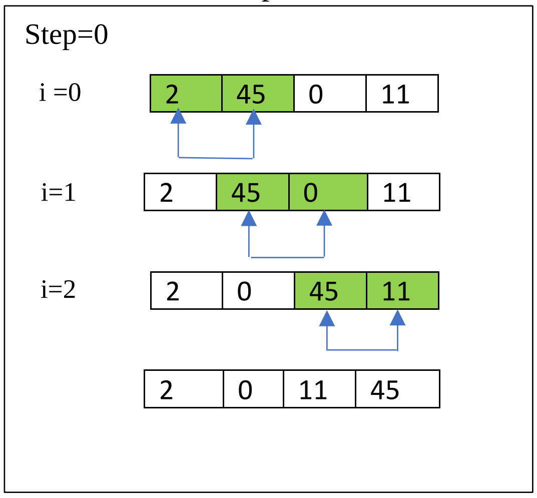
1. Introduction-Sorting Algorithm

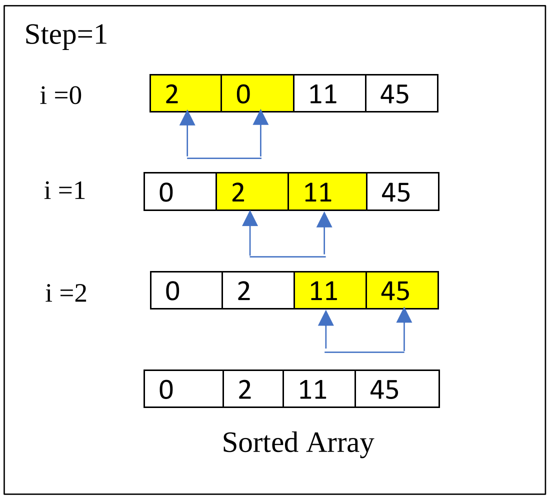
A Sorting Algorithm is used to rearrange a given array or list of elements according to a particular order. The most frequently used orders are ascending or descending orders. Among these sorting algorithms Bubble Sort is the simplest sorting algorithm. This sorting algorithm is comparison-based algorithm in which each pair of adjacent elements is compared and the elements are swapped if they are not in order.

Working of Bubble Sort
Suppose we are trying to sort the elements in ascending order.

- First Iteration (Compare and Swap)
  - Starting from the first index, compare the first and the second elements.
  - If the first element is greater than the second element, they are swapped.
  - Now, compare the second and the third elements. Swap them if they are not in order.
  - The above process goes on until the last element.

- Remaining Iteration
  - The same process goes on for the remaining iterations.
  - After each iteration, the largest element among the unsorted elements is placed at the end.
  - In each iteration, the comparison takes place up to the last unsorted element.
  - The array is sorted when all the unsorted elements are placed at their correct positions.





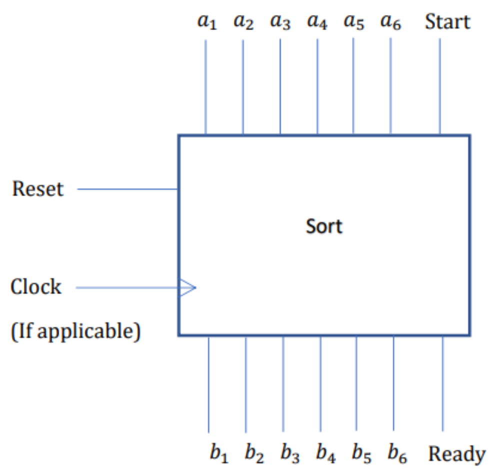
Complexity of Bubble Sort

Bubble Sort compares the adjacent elements.

Hence, the number of comparisons is (n-1) + (n-2) + (n-3) +.....+ 1 = n(n-1)/2

nearly equals to n^2. Hence, **Complexity:** $O(n^2)$

So in this mini project a sorting algorithm should be implemented to sort six (06) 16 bit integers a_1, a_2, a_3, a_4, a_5, a_6 and produce the output b_1, b_2, b_3, b_4, b_5, b_6 sorted in ascending order. As the sorting algorithm bubble sort can be used to implement this task.



a_1 a_2 a_3 a_4 a_5 a_6 Start
Reset
Sort
Clock
(If applicable)
b_1 b_2 b_3 b_4 b_5 b_6 Ready

Figure 01: Pin diagram for the module

2. Development of ASMD

- <u>Pseudo code</u>

```
output = b0,b1,b2,b3,b4,b5
input = a0,a1,a2,a3,a4,a5
while (1)
if (start)
temp_i=1
hold_a[1] = a0
hold_a[2] = a1
hold_a[3] = a2
hold_a[4] = a3
hold_a[5] = a4
hold_a[6] = a5
while (temp_i < 5)
temp_j=1
while (temp_j < 6)
if hold_a (temp_j) > hold_a (temp_j +1 )
temp=hold_a(temp_j)
```

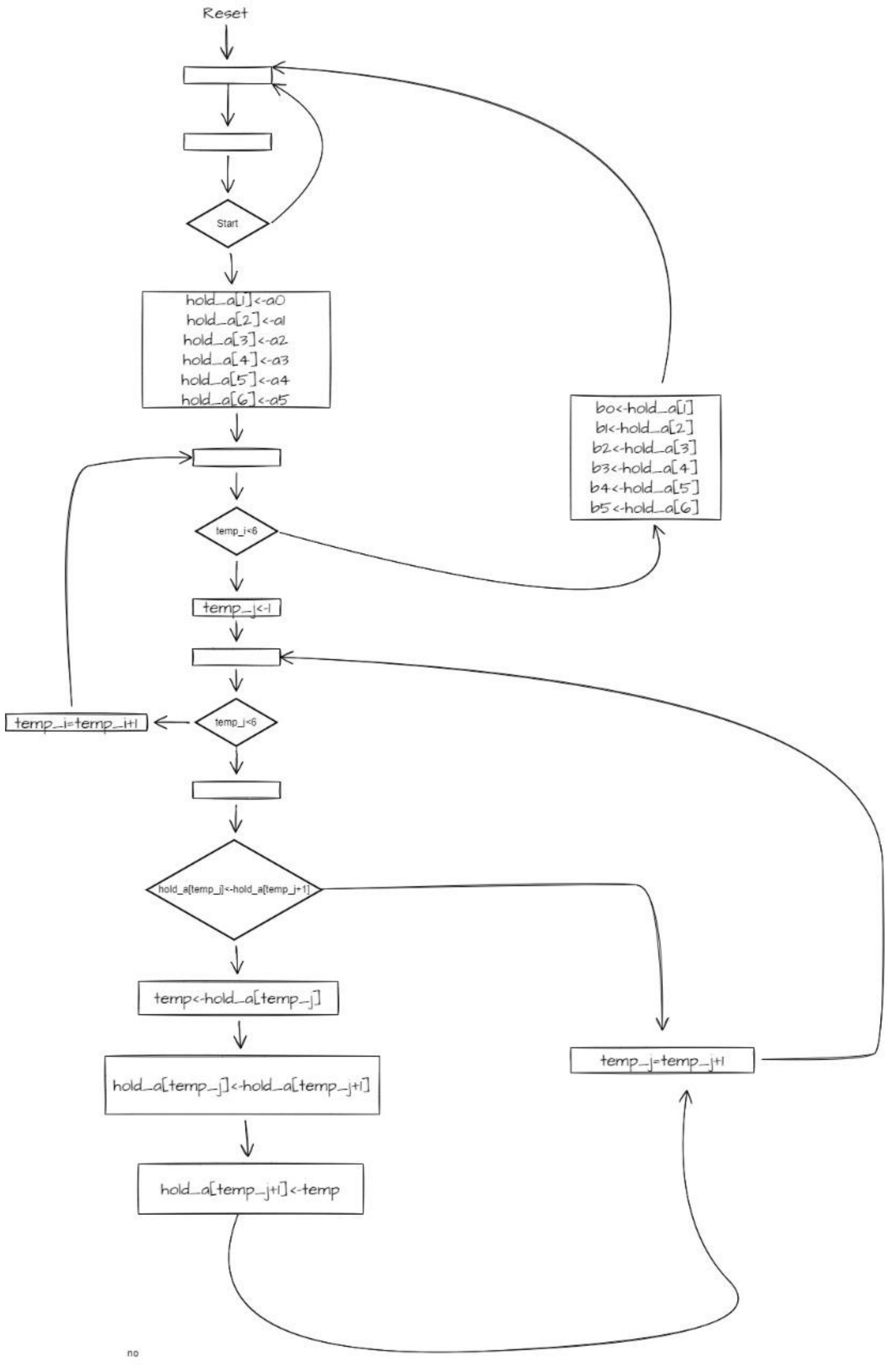
```
hold_a (temp_j)=hold_a (temp_j+1)
hold_a (tempj+1)=temp
end
temp_j=tempj+1
end
temp_i=temp_i+1
end
b0 = hold_a[1]
b1 = hold_a[2]
b2 = hold_a[3]
b3 = hold_a[4]
b4 = hold_a[5]
b5 = hold_a[6]
valid=1
end
end
```

- Conversion to register transfer operations

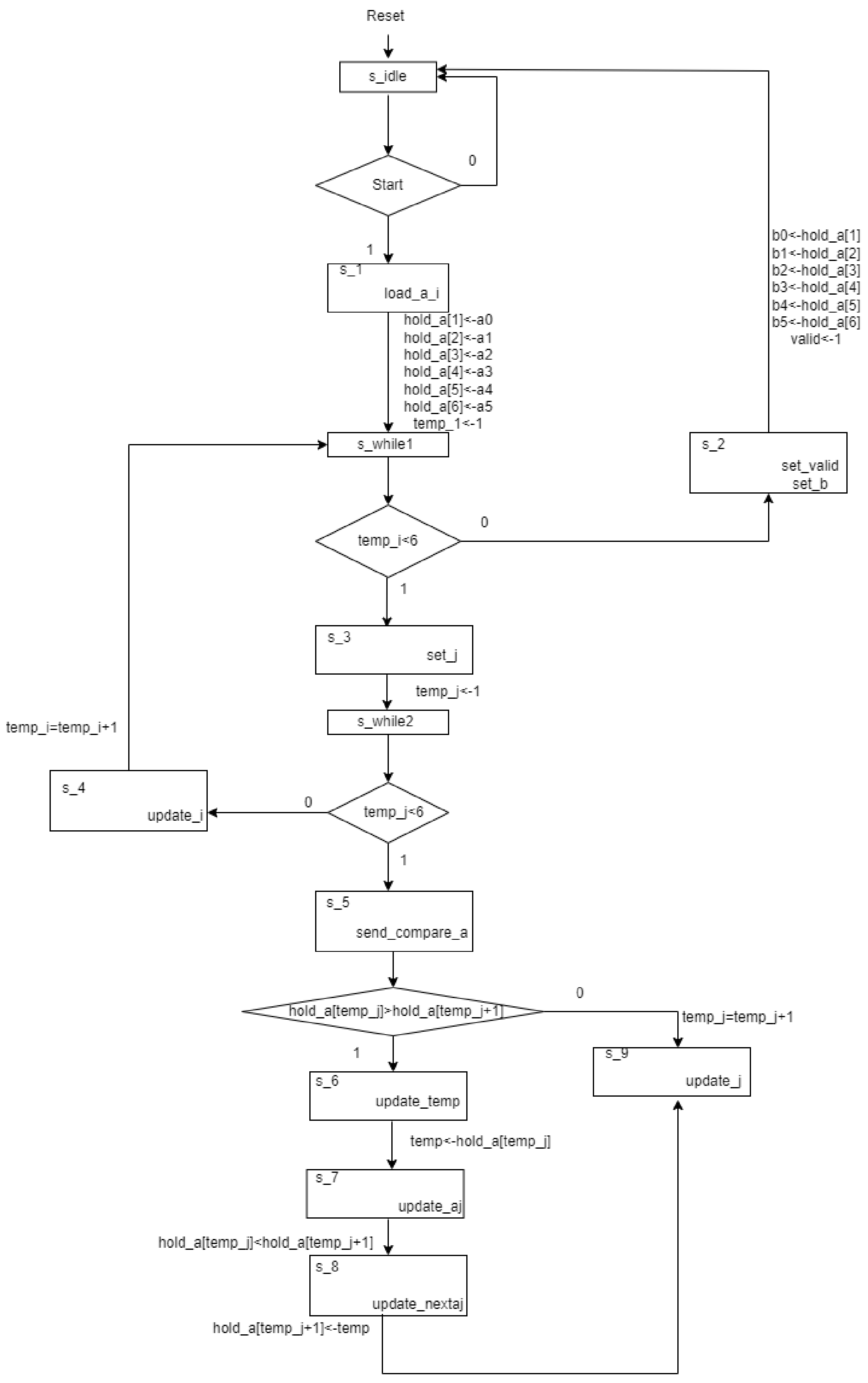
```
output ← b0,b1,b2,b3,b4,b5
input ← a0,a1,a2,a3,a4,a5
while (1)
if (start)
temp_i←1
hold_a[1] ←a0
hold_a[2] ←a1
hold_a[3] ←a2
hold_a[4] ←a3
hold_a[5] ←a4
hold_a[6] ←a5
while (temp_i < 5)
temp_j←1
while (temp_j < 6)
```

```
if hold_a (temp_j) > hold_a (temp_j +1 )
temp←hold_a(temp_j)
hold_a (temp_j) ←hold_a (temp_j+1)
hold_a (tempj+1) ←temp
end
temp_j←tempj+1
end
temp_i←temp_i+1
end
b0 ← hold_a[1]
b1 ← hold_a[2]
b2 ← hold_a[3]
b3 ← hold_a[4]
b4 ← hold_a[5]
b5 ← hold_a[6]
valid←1
end
end
```

- Flow chart


Reset
Start
hold_a[1]<-a0
hold_a[2]<-a1
hold_a[3]<-a2
hold_a[4]<-a3
hold_a[5]<-a4
hold_a[6]<-a5
temp_i<6
b0<-hold_a[1]
b1<-hold_a[2]
b2<-hold_a[3]
b3<-hold_a[4]
b4<-hold_a[5]
b5<-hold_a[6]
temp_j<-1
temp_i=temp_i+1
temp_j<6
hold_a[temp_j]<-hold_a[temp_j+1]
temp<-hold_a[temp_j]
hold_a[temp_j]<-hold_a[temp_j+1]
hold_a[temp_j+1]<-temp
temp_j=temp_j+1
no

- ASMD

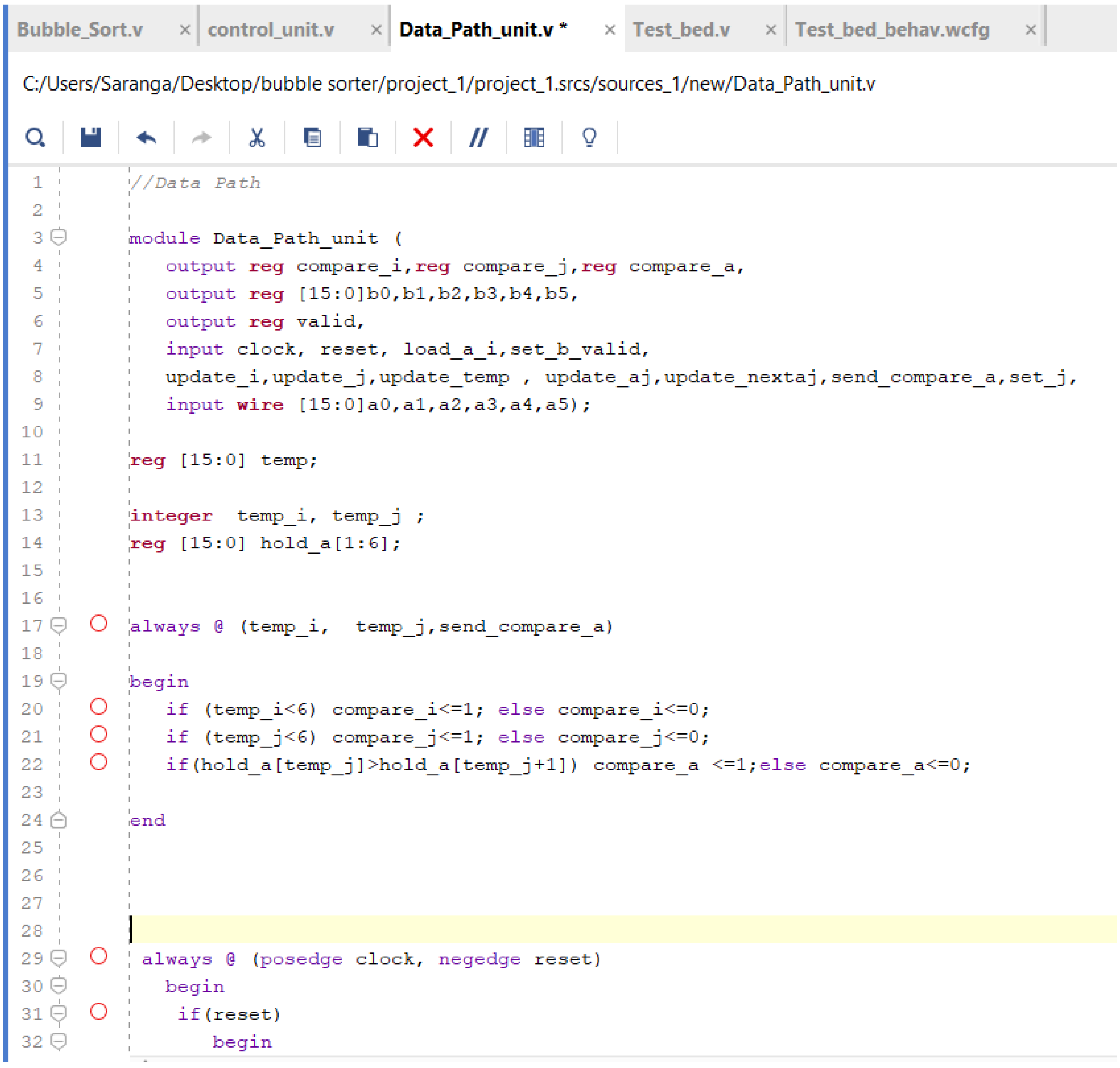


Reset
s_idle
Start
0
1
s_1
load_a_i
hold_a[1]<-a0
hold_a[2]<-a1
hold_a[3]<-a2
hold_a[4]<-a3
hold_a[5]<-a4
hold_a[6]<-a5
temp_1<-1
s_while1
temp_i<6
0
1
b0<-hold_a[1]
b1<-hold_a[2]
b2<-hold_a[3]
b3<-hold_a[4]
b4<-hold_a[5]
b5<-hold_a[6]
valid<-1
s_2
set_valid
set_b
s_3
set_j
temp_j<-1
s_while2
temp_i=temp_i+1
s_4
update_i
0
temp_j<6
1
s_5
send_compare_a
hold_a[temp_j]>hold_a[temp_j+1]
0
temp_j=temp_j+1
1
s_6
update_temp
s_9
update_j
temp<-hold_a[temp_j]
s_7
update_aj
hold_a[temp_j]<hold_a[temp_j+1]
s_8
update_nextaj
hold_a[temp_j+1]<-temp

3. Development of Testbed

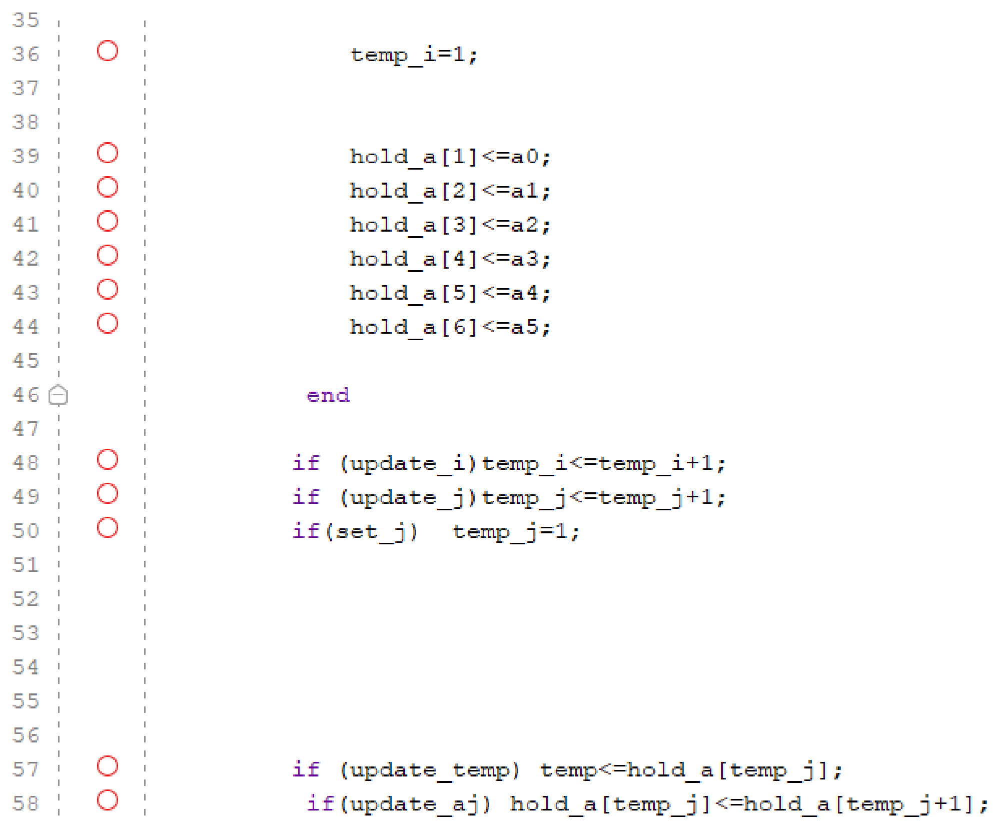
Xilinx Vivado webpack is used to design the Testbed of Sorting algorithm and Verilog is the Hardware description language. The Verilog codes are as follows:

- **Data Path**

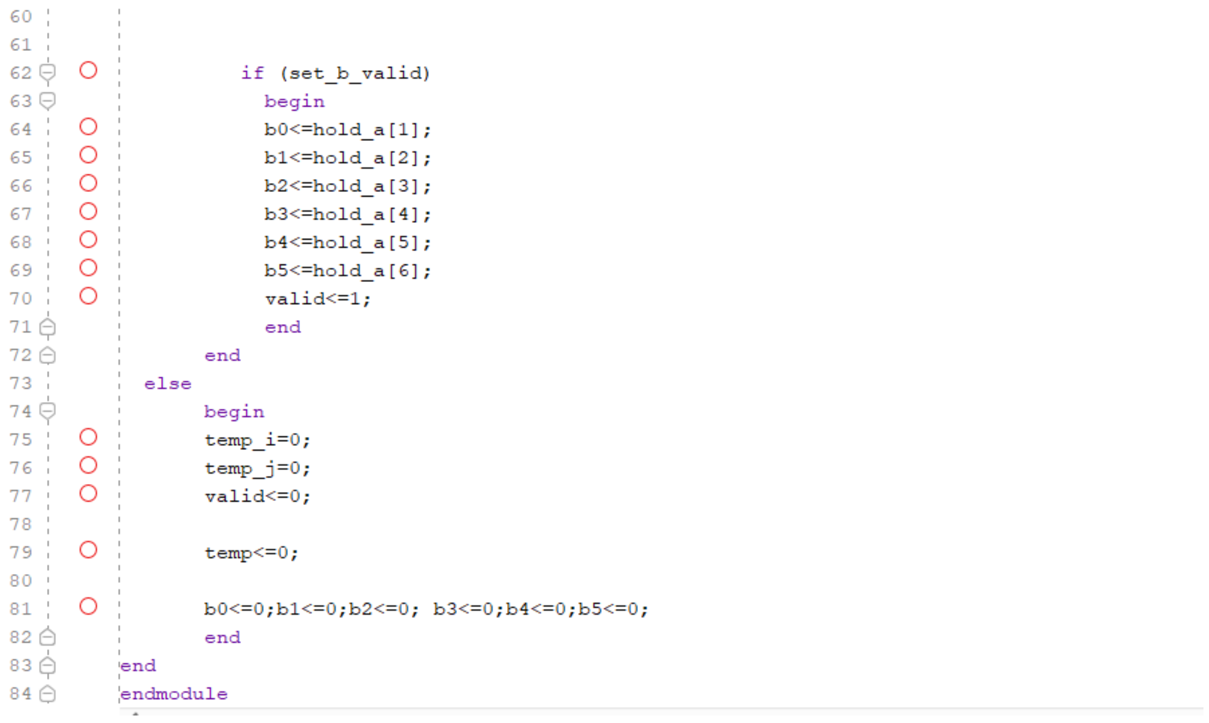


C:/Users/Saranga/Desktop/bubble sorter/project_1/project_1.srcs/sources_1/new/Data_Path_unit.v

```
//Data Path

module Data_Path_unit (
   output reg compare_i,reg compare_j,reg compare_a,
   output reg [15:0]b0,b1,b2,b3,b4,b5,
   output reg valid,
   input clock, reset, load_a_i,set_b_valid,
   update_i,update_j,update_temp , update_aj,update_nextaj,send_compare_a,set_j,
   input wire [15:0]a0,a1,a2,a3,a4,a5);

reg [15:0] temp;

integer  temp_i, temp_j ;
reg [15:0] hold_a[1:6];


always @ (temp_i,  temp_j,send_compare_a)

begin
   if (temp_i<6) compare_i<=1; else compare_i<=0;
   if (temp_j<6) compare_j<=1; else compare_j<=0;
   if(hold_a[temp_j]>hold_a[temp_j+1]) compare_a <=1;else compare_a<=0;

end



 always @ (posedge clock, negedge reset)
   begin
    if(reset)
       begin
```

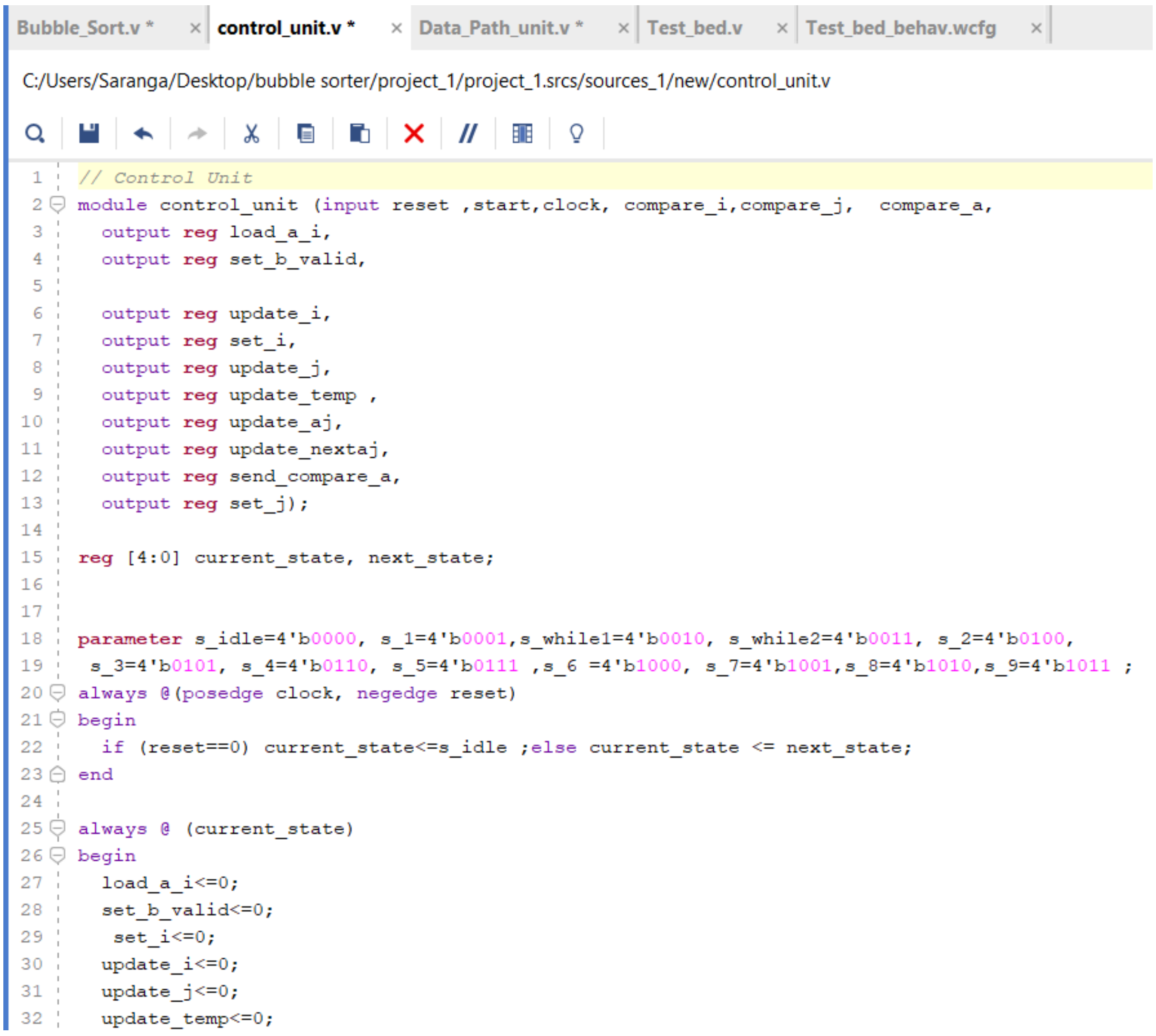


```
              temp_i=1;


              hold_a[1]<=a0;
              hold_a[2]<=a1;
              hold_a[3]<=a2;
              hold_a[4]<=a3;
              hold_a[5]<=a4;
              hold_a[6]<=a5;

           end

          if (update_i)temp_i<=temp_i+1;
          if (update_j)temp_j<=temp_j+1;
          if(set_j)  temp_j=1;






          if (update_temp) temp<=hold_a[temp_j];
           if(update_aj) hold_a[temp_j]<=hold_a[temp_j+1];
```



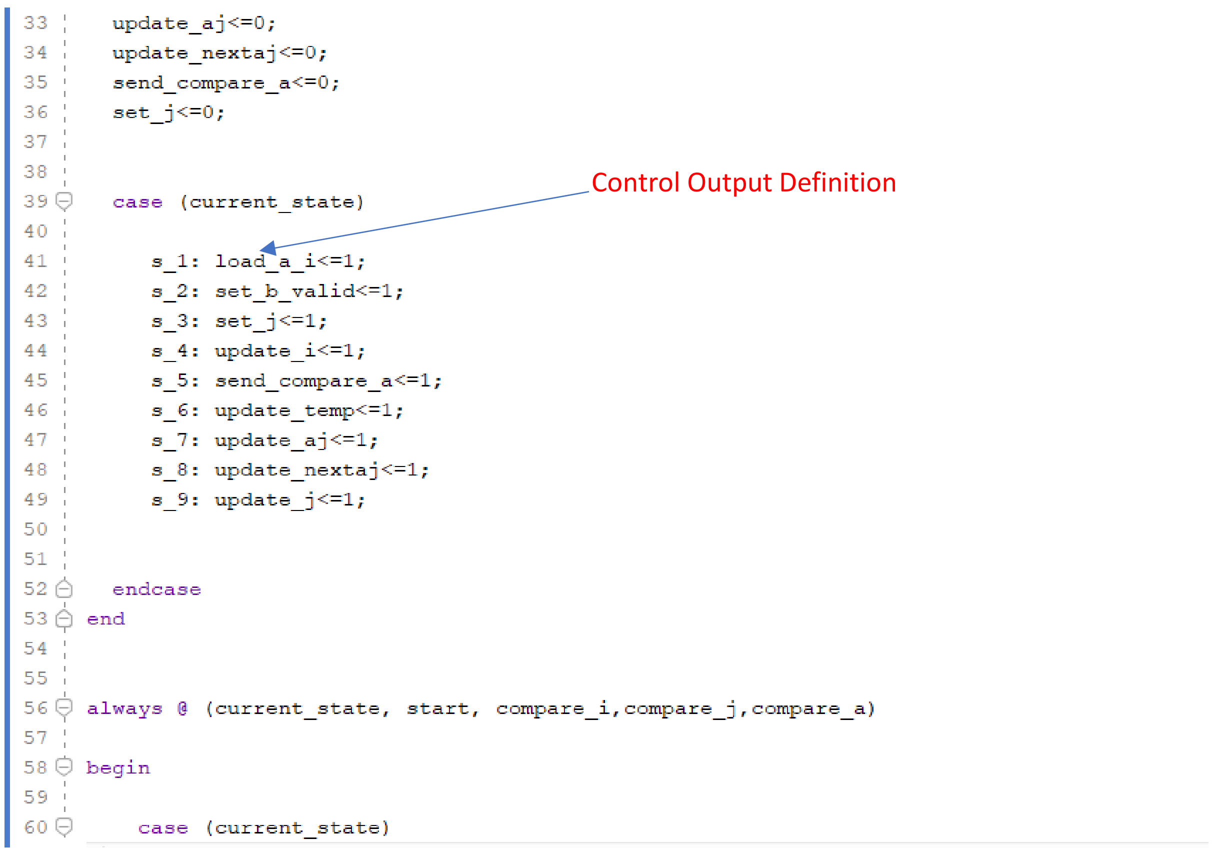
```


          if (set_b_valid)
            begin
            b0<=hold_a[1];
            b1<=hold_a[2];
            b2<=hold_a[3];
            b3<=hold_a[4];
            b4<=hold_a[5];
            b5<=hold_a[6];
            valid<=1;
            end
       end
  else
       begin
       temp_i=0;
       temp_j=0;
       valid<=0;

       temp<=0;

       b0<=0;b1<=0;b2<=0; b3<=0;b4<=0;b5<=0;
       end
end
endmodule
```

- **Control Unit**

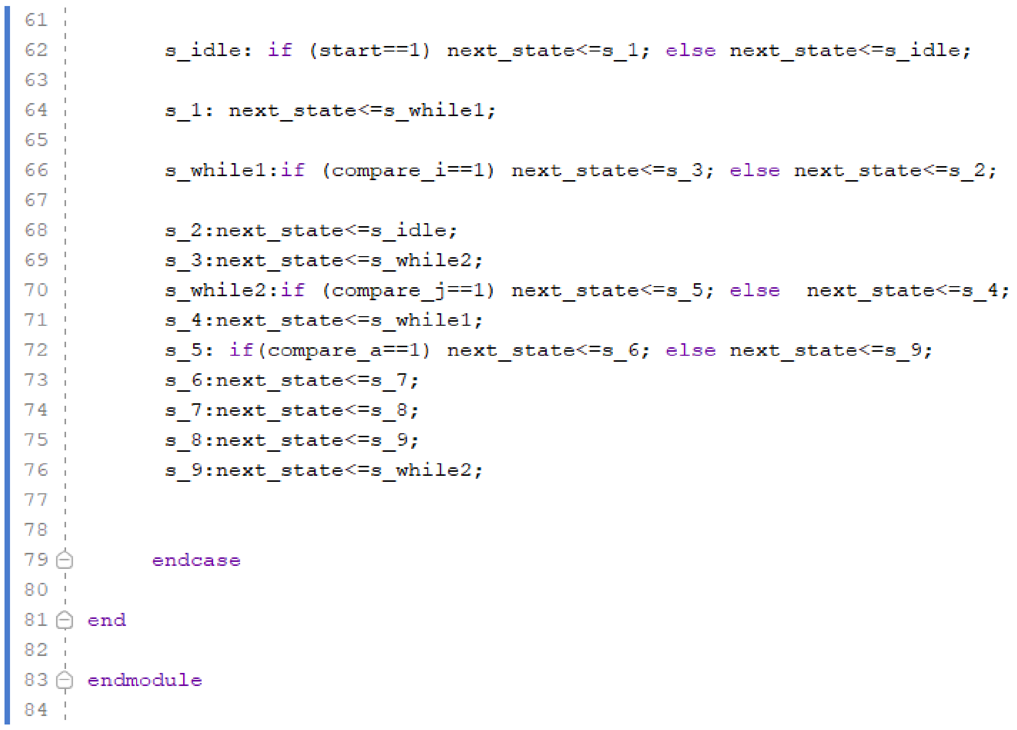


C:/Users/Saranga/Desktop/bubble sorter/project_1/project_1.srcs/sources_1/new/control_unit.v

```verilog
// Control Unit
module control_unit (input reset ,start,clock, compare_i,compare_j,  compare_a,
  output reg load_a_i,
  output reg set_b_valid,

  output reg update_i,
  output reg set_i,
  output reg update_j,
  output reg update_temp ,
  output reg update_aj,
  output reg update_nextaj,
  output reg send_compare_a,
  output reg set_j);

reg [4:0] current_state, next_state;


parameter s_idle=4'b0000, s_1=4'b0001,s_while1=4'b0010, s_while2=4'b0011, s_2=4'b0100,
 s_3=4'b0101, s_4=4'b0110, s_5=4'b0111 ,s_6 =4'b1000, s_7=4'b1001,s_8=4'b1010,s_9=4'b1011 ;
always @(posedge clock, negedge reset)
begin
  if (reset==0) current_state<=s_idle ;else current_state <= next_state;
end

always @ (current_state)
begin
  load_a_i<=0;
  set_b_valid<=0;
   set_i<=0;
  update_i<=0;
  update_j<=0;
  update_temp<=0;
```

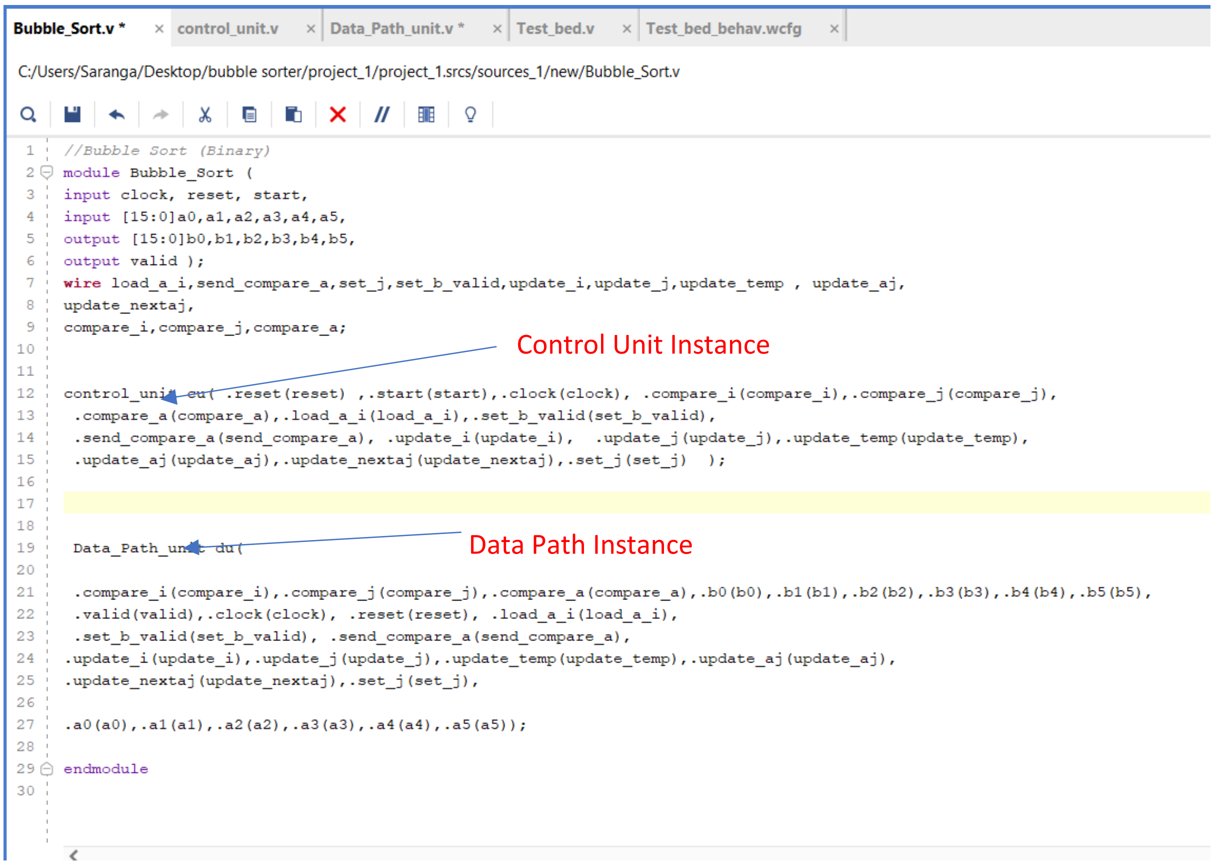


```
  update_aj<=0;
  update_nextaj<=0;
  send_compare_a<=0;
  set_j<=0;


  case (current_state)

     s_1: load_a_i<=1;
     s_2: set_b_valid<=1;
     s_3: set_j<=1;
     s_4: update_i<=1;
     s_5: send_compare_a<=1;
     s_6: update_temp<=1;
     s_7: update_aj<=1;
     s_8: update_nextaj<=1;
     s_9: update_j<=1;


  endcase
end


always @ (current_state, start, compare_i,compare_j,compare_a)

begin

    case (current_state)
```



```

      s_idle: if (start==1) next_state<=s_1; else next_state<=s_idle;

      s_1: next_state<=s_while1;

      s_while1:if (compare_i==1) next_state<=s_3; else next_state<=s_2;

      s_2:next_state<=s_idle;
      s_3:next_state<=s_while2;
      s_while2:if (compare_j==1) next_state<=s_5; else  next_state<=s_4;
      s_4:next_state<=s_while1;
      s_5: if(compare_a==1) next_state<=s_6; else next_state<=s_9;
      s_6:next_state<=s_7;
      s_7:next_state<=s_8;
      s_8:next_state<=s_9;
      s_9:next_state<=s_while2;


     endcase

end

endmodule

```

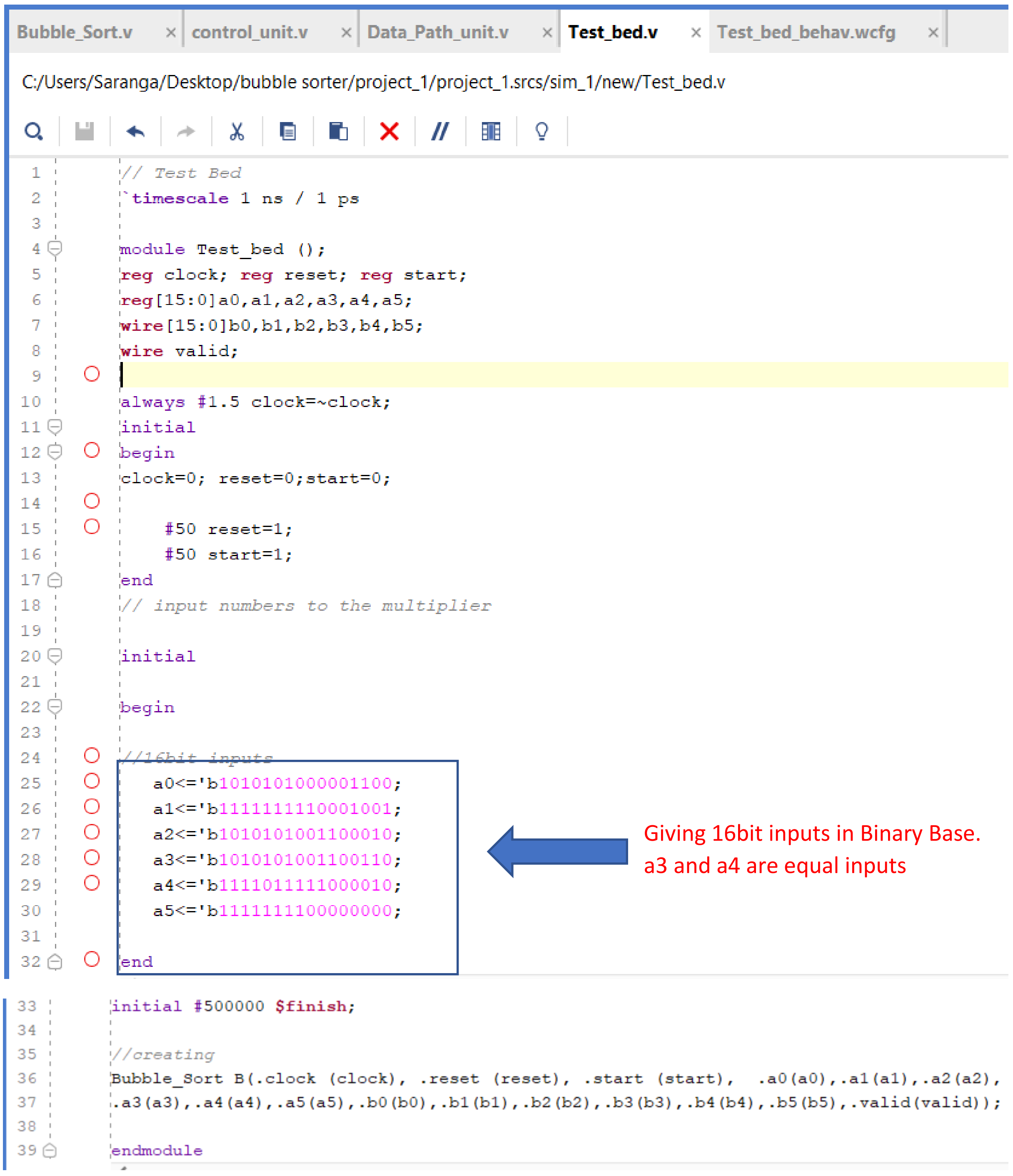
- **Bubble Sorter**



C:/Users/Saranga/Desktop/bubble sorter/project_1/project_1.srcs/sources_1/new/Bubble_Sort.v

```
//Bubble Sort (Binary)
module Bubble_Sort (
input clock, reset, start,
input [15:0]a0,a1,a2,a3,a4,a5,
output [15:0]b0,b1,b2,b3,b4,b5,
output valid );
wire load_a_i,send_compare_a,set_j,set_b_valid,update_i,update_j,update_temp , update_aj,
update_nextaj,
compare_i,compare_j,compare_a;


control_unit cu( .reset(reset) ,.start(start),.clock(clock), .compare_i(compare_i),.compare_j(compare_j),
 .compare_a(compare_a),.load_a_i(load_a_i),.set_b_valid(set_b_valid),
 .send_compare_a(send_compare_a), .update_i(update_i),  .update_j(update_j),.update_temp(update_temp),
 .update_aj(update_aj),.update_nextaj(update_nextaj),.set_j(set_j)  );



 Data_Path_unit du(

 .compare_i(compare_i),.compare_j(compare_j),.compare_a(compare_a),.b0(b0),.b1(b1),.b2(b2),.b3(b3),.b4(b4),.b5(b5),
 .valid(valid),.clock(clock), .reset(reset), .load_a_i(load_a_i),
 .set_b_valid(set_b_valid), .send_compare_a(send_compare_a),
.update_i(update_i),.update_j(update_j),.update_temp(update_temp),.update_aj(update_aj),
.update_nextaj(update_nextaj),.set_j(set_j),

.a0(a0),.a1(a1),.a2(a2),.a3(a3),.a4(a4),.a5(a5));

endmodule
```

- **Test Bed and Inputs**



C:/Users/Saranga/Desktop/bubble sorter/project_1/project_1.srcs/sim_1/new/Test_bed.v

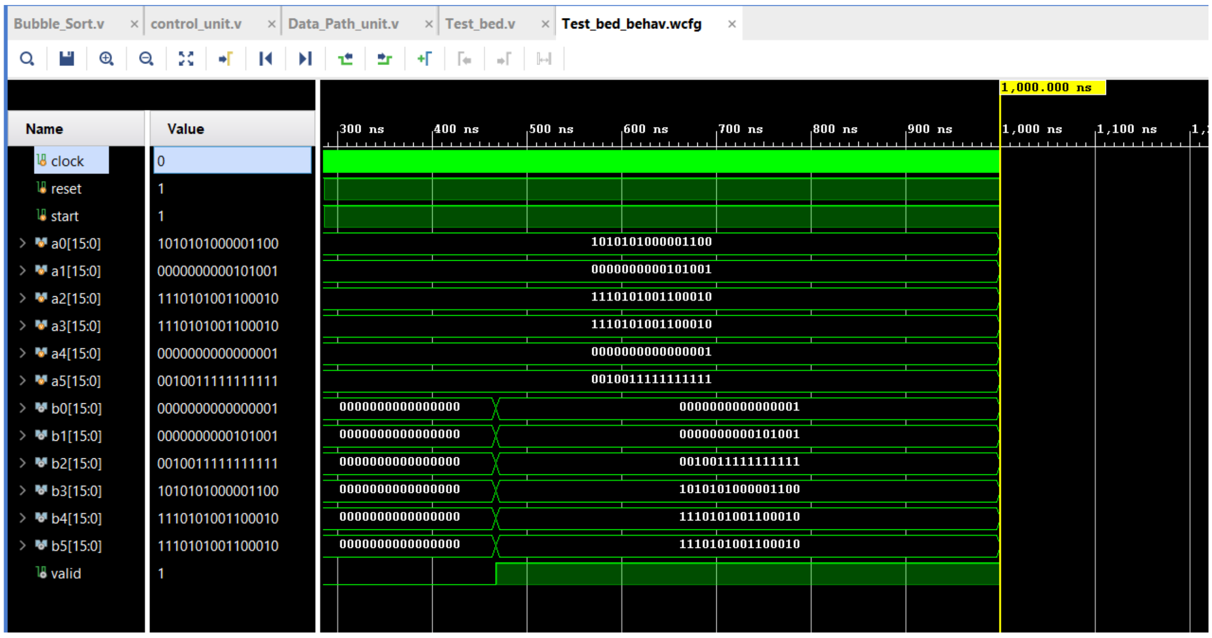
```verilog
// Test Bed
`timescale 1 ns / 1 ps

module Test_bed ();
reg clock; reg reset; reg start;
reg[15:0]a0,a1,a2,a3,a4,a5;
wire[15:0]b0,b1,b2,b3,b4,b5;
wire valid;

always #1.5 clock=~clock;
initial
begin
clock=0; reset=0;start=0;

    #50 reset=1;
    #50 start=1;
end
// input numbers to the multiplier

initial

begin
//16bit inputs
   a0<='b1010101000001100;
   a1<='b1111111110001001;
   a2<='b1010101001100010;
   a3<='b1010101001100110;
   a4<='b1111011111000010;
   a5<='b1111111100000000;

end
initial #500000 $finish;

//creating
Bubble_Sort B(.clock (clock), .reset (reset), .start (start),  .a0(a0),.a1(a1),.a2(a2),
.a3(a3),.a4(a4),.a5(a5),.b0(b0),.b1(b1),.b2(b2),.b3(b3),.b4(b4),.b5(b5),.valid(valid));

endmodule
```

Giving 16bit inputs in Binary Base.
a3 and a4 are equal inputs

- **<u>Results</u>**

Test Run 01

Input In Binary Numbers	In Decimal
A0 = 1010101000001100	43532
A1 = 0000000000101001	00041
A2 = 1110101001100010	60002
A3 = 1110101001100010	60002
A4 = 0000000000000001	00001
A5 = 0010011111111111	10239



Name	Value
clock	0
reset	1
start	1
a0[15:0]	1010101000001100
a1[15:0]	0000000000101001
a2[15:0]	1110101001100010
a3[15:0]	1110101001100010
a4[15:0]	0000000000000001
a5[15:0]	0010011111111111
b0[15:0]	0000000000000001
b1[15:0]	0000000000101001
b2[15:0]	0010011111111111
b3[15:0]	1010101000001100
b4[15:0]	1110101001100010
b5[15:0]	1110101001100010
valid	1

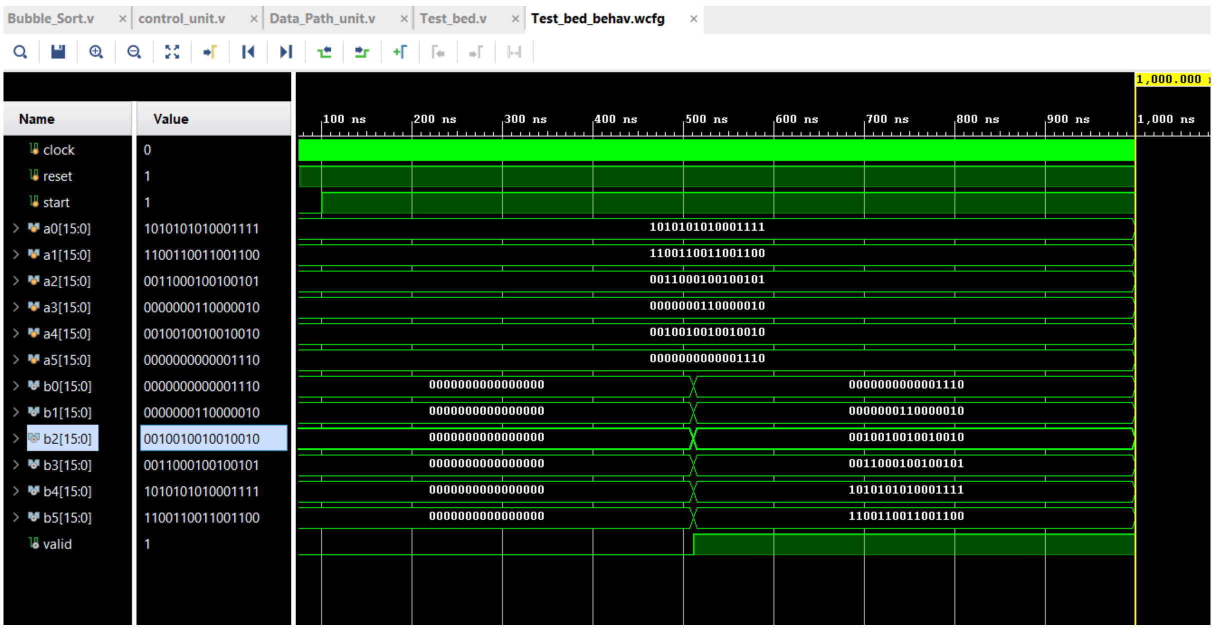
Output In Binary (from the output figure)	In Decimal
B0 = 0000000000000001	00001
B1 = 0000000000101001	00041
B2 = 0010011111111111	10239
B3 = 1010101000001100	43532
B4 = 1110101001100010	60002
B5 = 1110101001100010	60002

SORTED

Test Run 01 Successful

Test Run 02

Input In Binary Numbers	In Decimal
A0 = 1010101010001111	43663
A1 = 1100110011001100	52428
A2 = 0011000100100101	12581
A3 = 0000000110000010	00386
A4 = 0010010010010010	09362
A5 = 0000000000001110	00014



Name	Value
clock	0
reset	1
start	1
a0[15:0]	1010101010001111
a1[15:0]	1100110011001100
a2[15:0]	0011000100100101
a3[15:0]	0000000110000010
a4[15:0]	0010010010010010
a5[15:0]	0000000000001110
b0[15:0]	0000000000001110
b1[15:0]	0000000110000010
b2[15:0]	0010010010010010
b3[15:0]	0011000100100101
b4[15:0]	1010101010001111
b5[15:0]	1100110011001100
valid	1

Output In Binary (from the output figure)	In Decimal
B0 = 0000000000001110	00014
B1 = 0010010010010010	09362
B2 = 0000000110000010	00386
B3 = 0011000100100101	12581
B4 = 1010101010001111	43663
B5 = 1100110011001100	52428

SORTED

Test Run 02 Successful