EE3205 - Digital System Design

MINI PROJECT

GROUP MEMBERS - 19/ENG/024

19/ENG/058

19/ENG/074

19/ENG/119

SELECTION SORT ALGORITHM FOR MEMORY WRITING

1.Introduction

Selection sort is a basic sorting algorithm that works by repeatedly selecting the smallest element from the unsorted portion of an array and placing it at the beginning of a sorted portion. The algorithm starts by assuming the first element of the array as the smallest, then compares it with the rest of the elements of the array. If it finds a smaller element, it swaps the smallest element with the element that is smaller. This process is repeated for all the remaining unsorted elements of the array until the entire array is sorted. The algorithm has a time complexity of O(n^2) which makes it less efficient for larger arrays compared to other sorting algorithms like quicksort and merge sort.

Visualizing the selection sort algorithm,

original array,

15	30	25	10	35	20	45	40

Find minimum value,

15	30	25	10	35	20	45	40

Swap with current position,

10	30	25	15	35	20	45	40

Move to next,

10	30	25	15	35	20	45	40

In our project, we are utilizing the selection sort algorithm for memory management. We are doing this by taking multiple files and their sizes into consideration. Then, we use the selection sort algorithm to sort the files in ascending order. Once the sorting is done, the files can be saved to the drive in their sorted order.

Figure 1: getting size of any file as input

Selection sort algorithm is implemented to sort six (6) 16bit integers, such as a0, a1, a2, a3, a4, a5 and produce the output, b0, b1, b2, b3, b4, b5 which is sorted in ascending order. The pin diagram for the module is as figure 2. Once start pin become logic-1 the operation begins and once the sorting is completed, ready pin become logic-1.

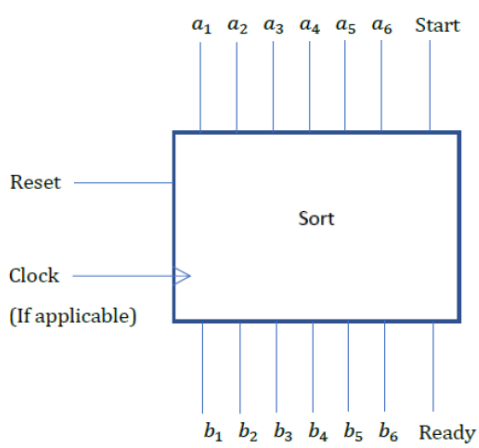


Figure 2: Pin diagram

2.Development of ASMD

- **Step 1: Pseudo code**

```
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
                temp_min=1
                while (temp_i < 7)
                        temp_j= temp_i
                        while (temp_j < 7)
                                if hold_a (temp_i) >= hold_a (temp_j )
                                        temp_min =hold_a(temp_j)
                                        temp_index= temp_j
                                end
                                temp_j=tempj+1
                        end
                        hold_a (temp_ index)=hold_a (temp_i)
                        hold_a (temp_i)= temp_min
                        temp_i=temp_i+1
                end
                b0 = hold_a[1]
                b1 = hold_a[2]
```

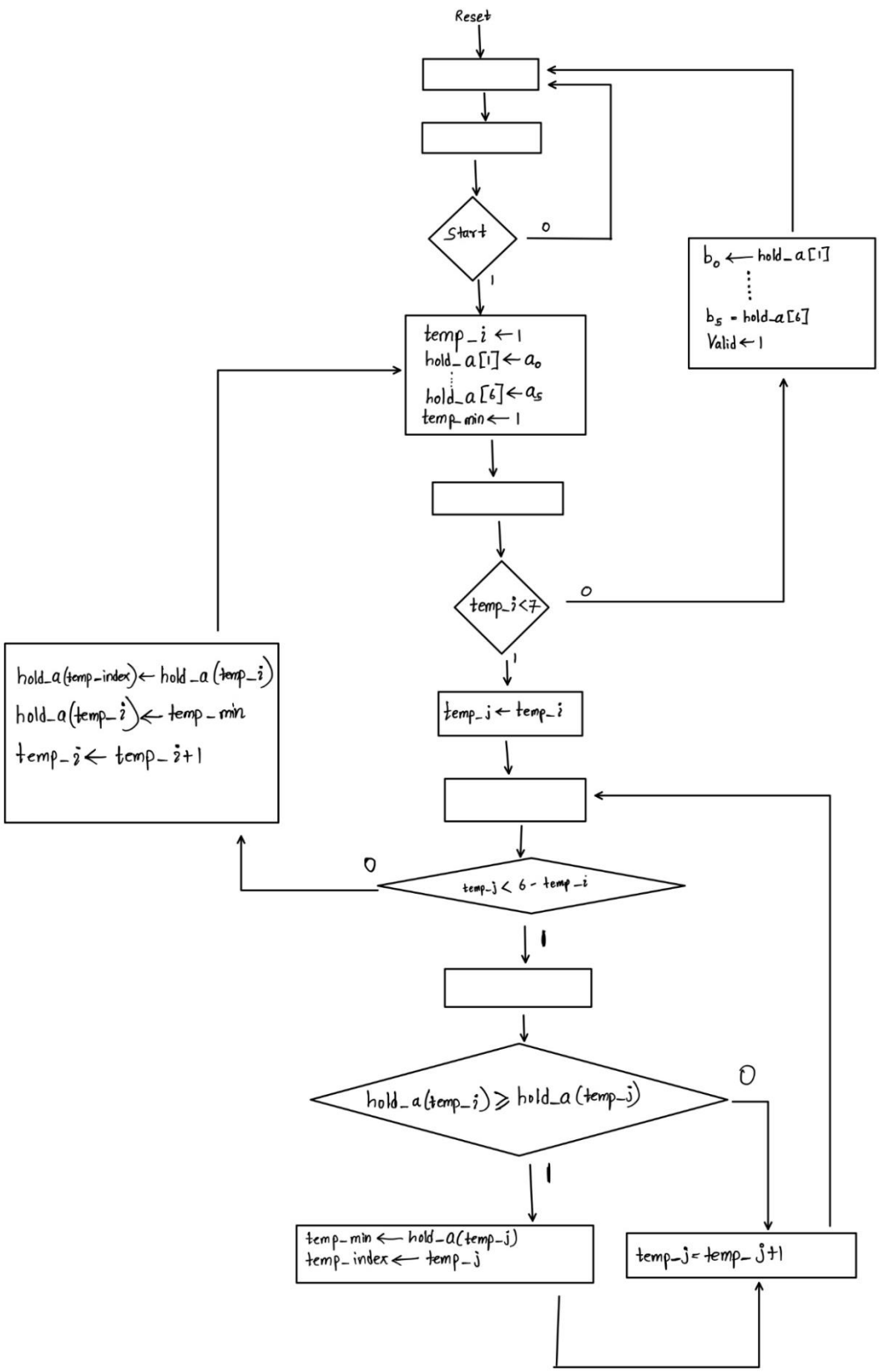
```
                b2 = hold_a[3]
                b3 = hold_a[4]
                b4 = hold_a[5]
                b5 = hold_a[6]
                valid=1
        end
end
```

- **Step 2: register transfer operations**

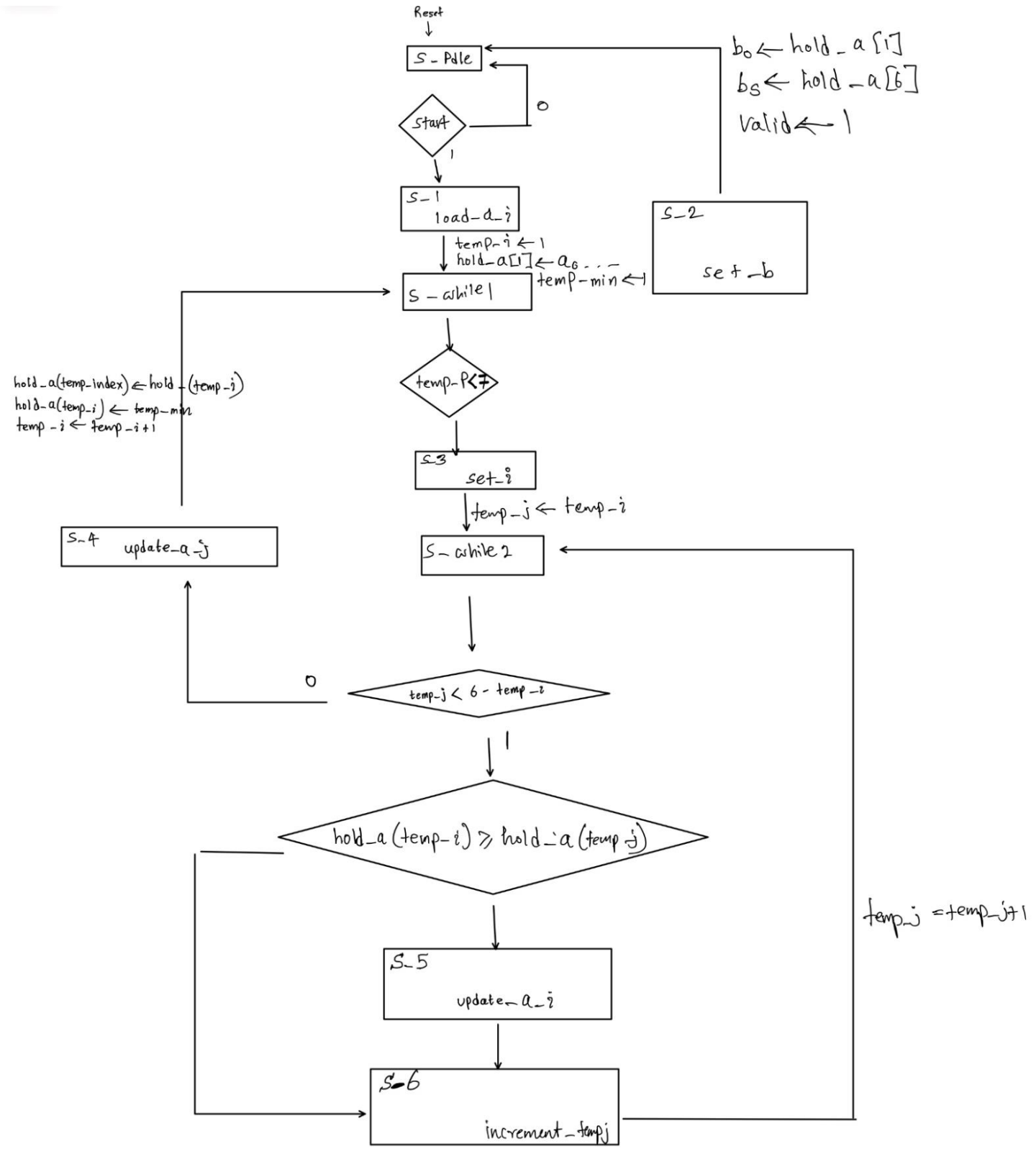
```
output ← b0,b1,b2,b3,b4,b5
input ← a0,a1,a2,a3,a4,a5
while (1)
        if (start)
                temp_i←1
                hold_a[1] ← a0
                hold_a[2] ← a1
                hold_a[3] ← a2
                hold_a[4] ← a3
                hold_a[5] ← a4
                hold_a[6] ← a5
                temp_min←1
                while (temp_i < 7)
                        temp_j← temp_i
                        while (temp_j < 7)
                                if hold_a (temp_i) >= hold_a (temp_j )
                                        temp_min ←hold_a(temp_j)
                                        temp_index← temp_j
                                end
                                temp_j←tempj+1
```

```
                end
                hold_a (temp_ index) ←hold_a (temp_i)
                hold_a (temp_i) ← temp_min
                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
```

- **Step 3: Flow chart**



- **Step 4: ASMD**


Reset
S - Idle
Start
0
1
S - 1
load - a - i
temp - i ← 1
hold - a[1] ← a6 . . .
temp - min ←
S - while 1
temp - i < 7
S - 2
set - b
b0 ← hold - a [1]
b5 ← hold - a [6]
Valid ← 1
hold - a(temp - index) ← hold - (temp - i)
hold - a(temp - i) ← temp - min
temp - i ← temp - i + 1
S - 3
set - i
temp - j ← temp - i
S - 4
update - a - j
S - while 2
0
temp - j < 6 - temp - i
1
hold - a (temp - i) ≥ hold - a (temp j)
temp - j = temp - j + 1
S - 5
update - a - i
S - 6
increment - temp j

3. Development of Testbed

For the selection sorting module and the test bench, Xilinx Vivado webpack is used with Verilog as the hardware description language (HDL). HDL model hierachy is in figure 3.

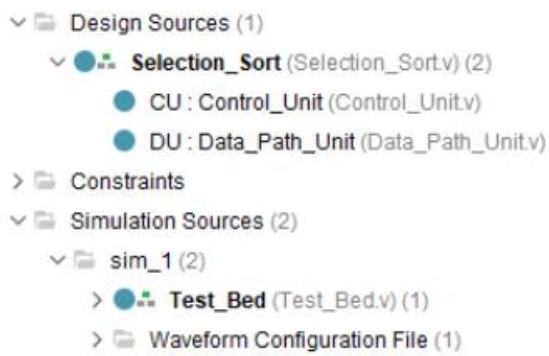
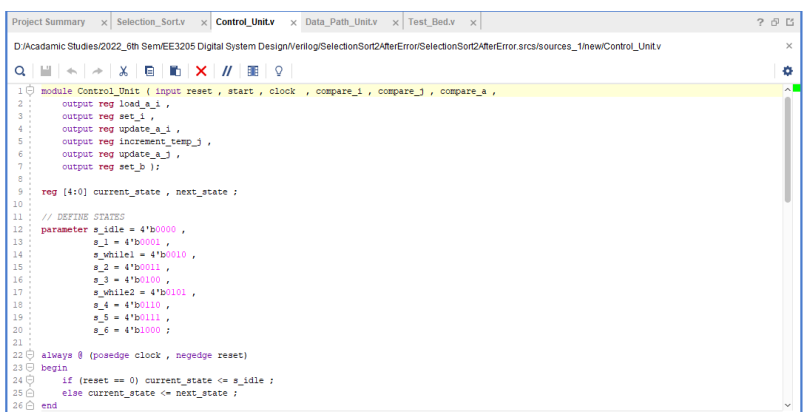


Figure 3: HDL model hierarchy

The codes for Control unit, Data path unit, Selection sort and testbed implementations are as follows.

• Designing of the Control Unit of Selection sort algorithm




```
module Control_Unit ( input reset , start , clock  , compare_i , compare_j , compare_a ,
    output reg load_a_i ,
    output reg set_i ,
    output reg update_a_i ,
    output reg increment_temp_j ,
    output reg update_a_j ,
    output reg set_b );

reg [4:0] current_state , next_state ;

// DEFINE STATES
parameter s_idle = 4'b0000 ,
          s_1 = 4'b0001 ,
          s_while1 = 4'b0010 ,
          s_2 = 4'b0011 ,
          s_3 = 4'b0100 ,
          s_while2 = 4'b0101 ,
          s_4 = 4'b0110 ,
          s_5 = 4'b0111 ,
          s_6 = 4'b1000 ;

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 <= 0 ;
    set_i <= 0 ;
    update_a_i <= 0 ;
    increment_temp_j <= 0 ;
    update_a_j <= 0 ;

    case (current_state)
        s_1 : load_a_i <= 1 ;
        s_2 : set_b <= 1 ;
        s_3 : set_i <= 1 ;
        s_4 : update_a_j <= 1 ;
        s_5 : update_a_i <= 1 ;
        s_6 : increment_temp_j <= 1 ;
    endcase
end
```

```
// STATE CHANGING
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) begin
                    if (compare_a == 1) next_state <= s_5 ;
                    else next_state <= s_6 ;
                   end
                   else next_state <= s_4 ;

        s_4 : next_state <= s_while1 ;
        s_5 : next_state <= s_6 ;
        s_6 : next_state <= s_while2 ;
    endcase
end

endmodule
```

In the control unit, first define inputs and outputs of Control Unit.

Then, initialize all states (s_1, s_2….) recognized in ASMD in binary format.

Finally, by looking at ASMD, define how states are changed based on current state. Based on current state next state will be determined.

- Designing of the Data Path Unit of Selection sort algorithm

Project Summary | Selection_Sort.v | Control_Unit.v | Data_Path_Unit.v | Test_Bed.v

D:/Acadamic Studies/2022_6th Sem/EE3205 Digital System Design/Verilog/SelectionSort2AfterError/SelectionSort2AfterError.srcs/sources_1/new/Data_Path_Unit.v

```
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_i , update_a_i , increment_temp_j , update_a_j , set_b ,
    input wire [15:0]  a0 , a1 , a2 , a3 , a4 , a5 ) ;

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

// GENERATE COMPARE SIGNALS
always @ (temp_i , temp_j )
begin
    if (temp_i < 7) compare_i <= 1 ; else compare_i <= 0 ;
    if (temp_j < 7) compare_j <= 1 ; else compare_j <= 0 ;
    if (hold_a[temp_i] >= hold_a[temp_j]) compare_a <= 1 ; else compare_a <= 0 ;
end

always @ (posedge clock , negedge reset)
begin
    if (reset) begin
        if (load_a_i) begin
            temp_i = 1 ;
            temp_min = 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 (set_i) begin
            temp_j = temp_i ;
        end

        if (update_a_i) begin
            temp_min <= hold_a[temp_j] ;
            temp_index <= temp_j ;
        end

        if (increment_temp_j) temp_j <= temp_j +1 ;

        if (update_a_j) begin
            hold_a[temp_index] <= hold_a[temp_i] ;
            hold_a[temp_i] <= temp_min ;
            temp_i <= temp_i +1 ;
        end
```

```
        if (set_b)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_j = 0 ;
        temp_i = 0 ;
        temp_index = 0 ;
        valid <= 0 ;
        b0 <= 0 ;
        b1 <= 0 ;
        b2 <= 0 ;
        b3 <= 0 ;
        b4 <= 0 ;
        b5 <= 0 ;
    end
end
endmodule
```

In the Data Path Unit aloso, first define inputs and outputs.

Next, generates compare signals.

Afterthat, define all the operations that need to be excecuted when the cintrol signal is issue.

- Designing of the Selection sort algorithm

Project Summary | Selection_Sort.v | Control_Unit.v | Data_Path_Unit.v | Test_Bed.v

D:/Acadamic Studies/2022_6th Sem/EE3205 Digital System Design/Verilog/SelectionSort2AfterError/SelectionSort2AfterError.srcs/sources_1/new/Selection_Sort.v

```
`timescale 1ns / 1ps

module Selection_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 compare_i ,  compare_j ,  compare_a ,
load_a_i , set_i , update_a_i , increment_temp_j , update_a_j , set_b ;


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_i(set_i) ,
.update_a_i(update_a_i) , .increment_temp_j(increment_temp_j) , .update_a_j(update_a_j) ,
.set_b(set_b)
) ;

Data_Path_Unit DU (
.reset(reset) , .clock(clock) , .b0(b0) , .b1(b1) , .b2(b2) , .b3(b3) , .b4(b4) , .b5(b5) ,
.compare_i(compare_i) , .compare_j(compare_j) , .compare_a(compare_a) , .load_a_i(load_a_i) , .set_i(set_i) ,
.update_a_i(update_a_i) , .increment_temp_j(increment_temp_j) , .update_a_j(update_a_j) ,
.set_b(set_b) , .a0(a0) , .a1(a1) , .a2(a2) , .a3(a3) , .a4(a4) , .a5(a5)
) ;

endmodule
```

In selection sort, instances of Control Unit and Data Path Unit are created. Instance created for Contol Unit is CU and instance created for Data Path Unit is DU.

- Designing of the TestBed of Selection sort algorithm

Project Summary | Selection_Sort.v | Control_Unit.v | Data_Path_Unit.v | Test_Bed.v

D:/Acadamic Studies/2022_6th Sem/EE3205 Digital System Design/Verilog/SelectionSort2AfterError/SelectionSort2AfterError.srcs/sim_1/new/Test_Bed.v

```
`timescale 1ns / 1ps

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

    initial
    begin
        a0 <= 10 ;
        a1 <= 10 ;
        a2 <= 2 ;
        a3 <= 110 ;
        a4 <= 85 ;
        a5 <= 240 ;

    end

    initial #500000 $finish ;

    Selection_Sort SS(
    .reset(reset) , .start(start) , .clock(clock) ,
    .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)
    );
```

Start and reset clock signal are generated here.

Inputs for the sort are given as a initial block. (ex: a0 <= 10; ….)

Finally create instance of Selection Sort as SS.

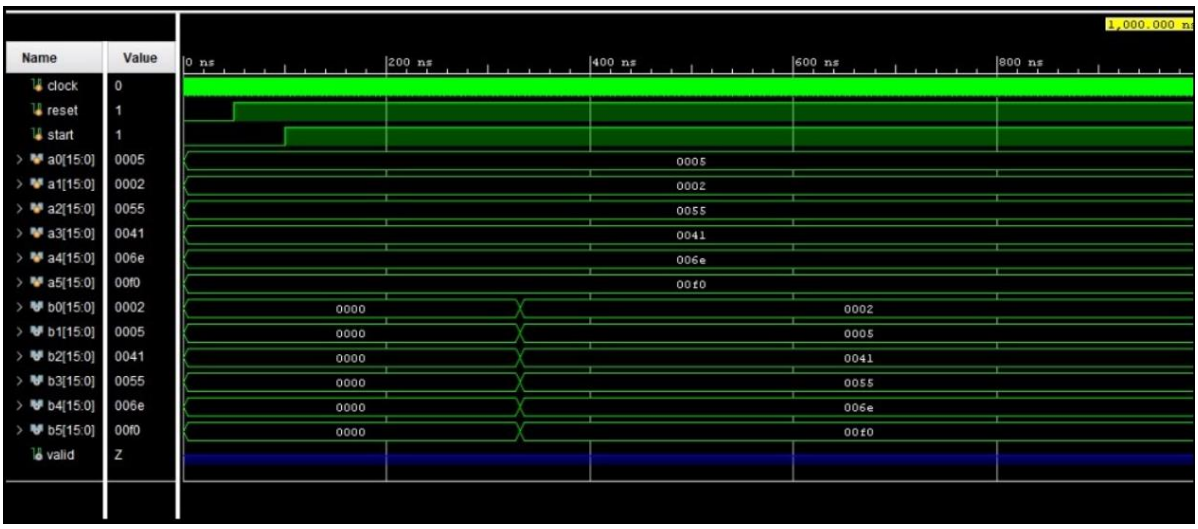
4. Results

Finally, the six 16bits input integers could be sorted in ascending order from this implementation and the obtained results are given below.

Simulation results

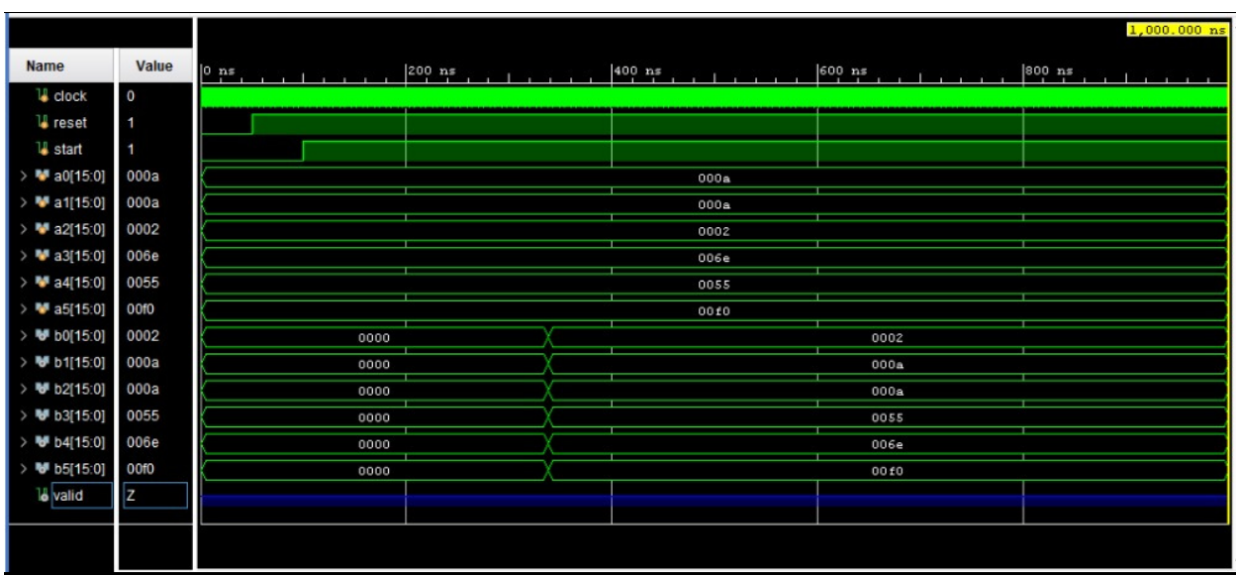
a. For different inputs

Inputs: 25, 10, 2, 110, 240, 85
Output: 2, 10,25,85,110,240



b. For same inputs

Inputs: 10, 10, 2, 110, 240, 85
Output: 2, 10,10,85,110,240



Schematic diagram