Jan. 1, 2014

COMPUTER ENGINEERING DEPARTMENT

COE 405

DESIGN & MODELING OF DIGITAL SYSTEMS

Final Exam

First Semester (131)

Time: 7:00-10:00 PM

OPEN BOOK EXAM

Student Name : _KEY_____

Student ID. : _____

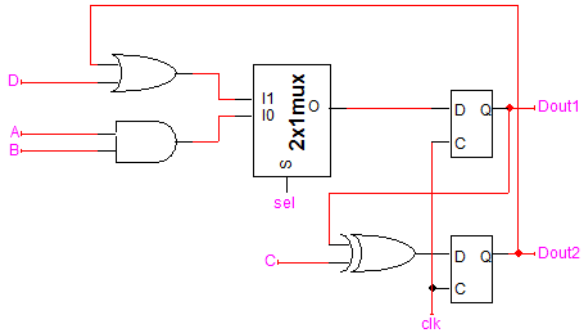
Question	Max Points	Score
Q1	20	
Q2	14	
Q3	13	
Q4	13	
Q5	40	
Total	100	

Dr. Aiman El-Maleh

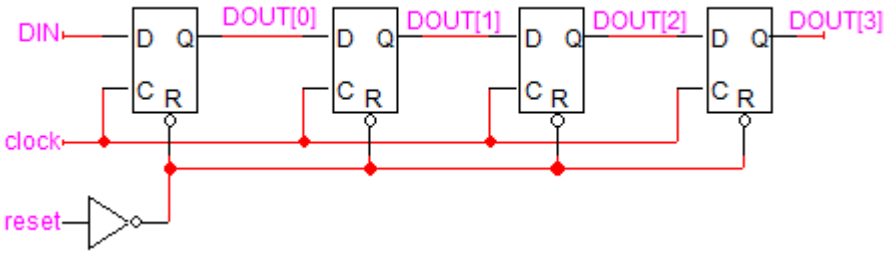
[20 Points]

(Q1) Determine possible circuits that will be synthesized by each of the following modules:

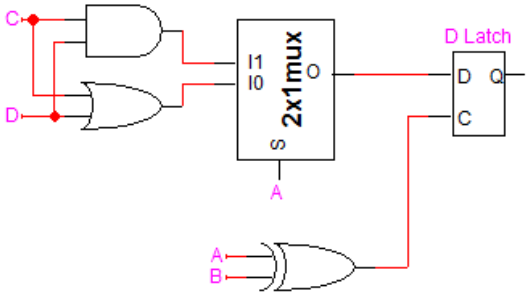
(i)
```
module FinalQ1_1 (output reg Dout1, Dout2, input A, B, C, D, sel, clk);
  always @ (posedge clk) begin
      Dout1 <= A & B; Dout2 <= Dout1 ^ C;
      if (sel) Dout1 <= Dout2 | D;
  end
endmodule
```



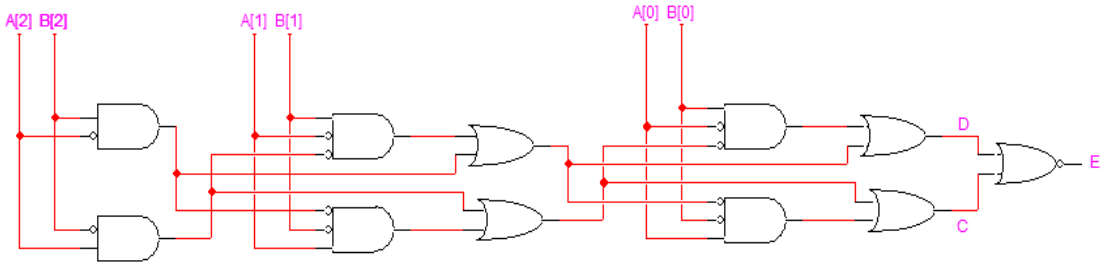
(ii)
```
module FinalQ1_2 (output reg [3:0] DOUT, input DIN, clock, reset);
   always @ (posedge clock,  posedge reset) begin
       if (reset) DOUT <= 0;
       else begin
                    DOUT[0]<=DIN;
                    DOUT[1]<=DOUT[0];
                    DOUT[2]<=DOUT[1];
                    DOUT[3]<=DOUT[2];
       end
   end
endmodule
```



(iii)
```
module FinalQ1_3 (output reg E, input A, B, C, D);

always @(A,B, C, D) begin
        case ({A,B})
        2'b10:  E = C & D;
        2'b01:  E = C | D;
        endcase
end
endmodule
```

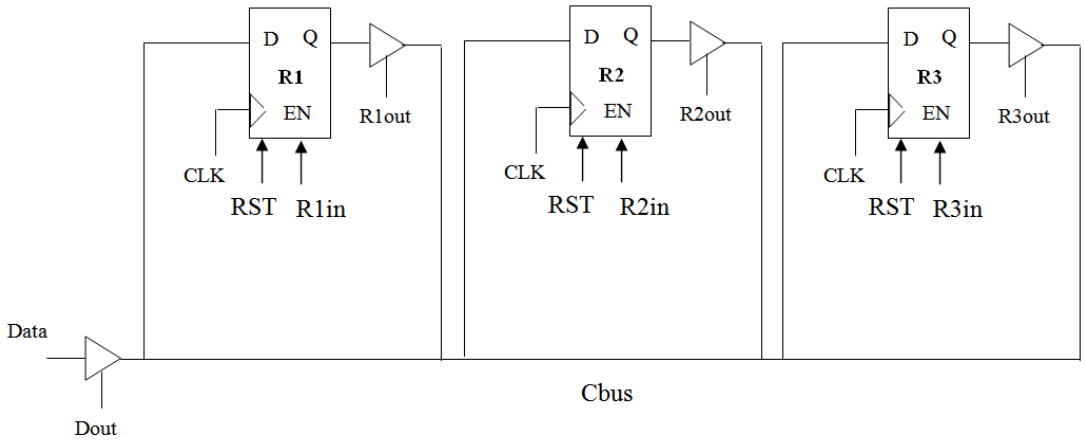


(iv)
```
module FinalQ1_4 #(parameter n=3)(output  reg C, D, output E, input [n-1:0] A, B);
  integer k;
  assign E = ~(C | D);
  always @ (A, B) begin
    C = 0; D = 0;
    for (k=n-1; k>=0; k=k-1) begin
      C = C | A[k] & ~B[k] & ~D;
      D = D | ~A[k] & B[k] & ~C;
    end
  end
endmodule
```



(v)

```
module FinalQ1_5 #(parameter n=4) (output reg [n-1:0] R1, R2, R3,  input CLK,
RST, R1in, R2in, R3in, R1out, R2out, R3out, Dout, input [n-1:0] Data);
  wire [n-1:0] Cbus;
  always @ (posedge CLK, posedge RST) begin
      if (RST) begin R1 <= 0; R2 <= 0; R3 <= 0;   end
      else begin
              if (R1in) R1 <= Cbus;
              if (R2in) R2 <= Cbus;
              if (R3in) R3 <= Cbus;
      end
  end
  assign Cbus = R1out? R1:{n{1'bz}};
  assign Cbus = R2out? R2:{n{1'bz}};
  assign Cbus = R3out? R3:{n{1'bz}};
  assign Cbus = Dout? Data:{n{1'bz}};
endmodule
```



[14 Points]

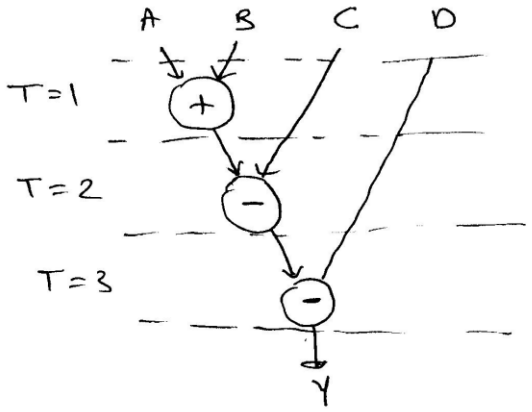
(Q2) It is required to design a circuit to compute the equation Y=(A+B)-(C+D) using only a single adder. Assume that the 4-bit inputs will hold their values until the end of the operation. Also, assume that the inputs will be ready when a **Start** input is asserted. Assume that a **Done** signal will be set when the result is ready and the result will remain held until the next Start operation.

(i) Show a schedule of the operations with minimum latency (i.e., clock cycles) satisfying the area constraints.

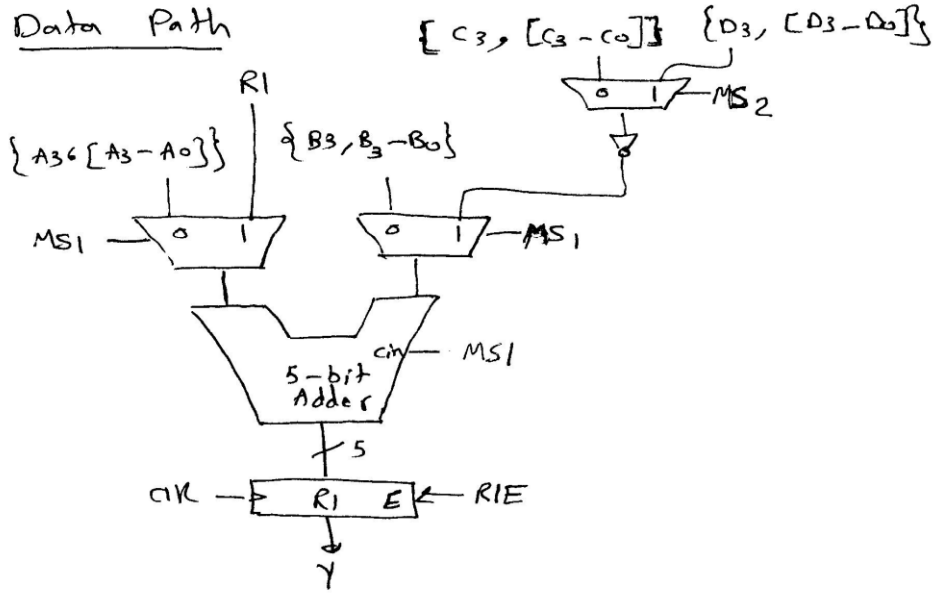
(ii) Show the DataPath design of your circuit indicating all the control signals and the adder size.

(iii) Show the ASMD diagram of your control unit.

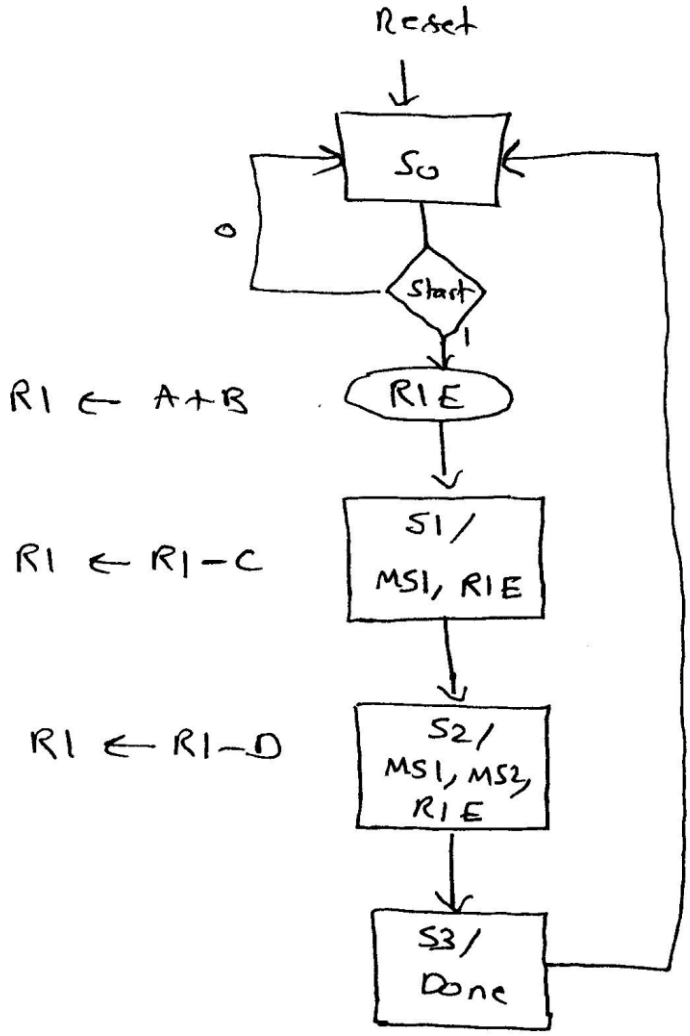
(i) Schedule :



(ii) Data Path

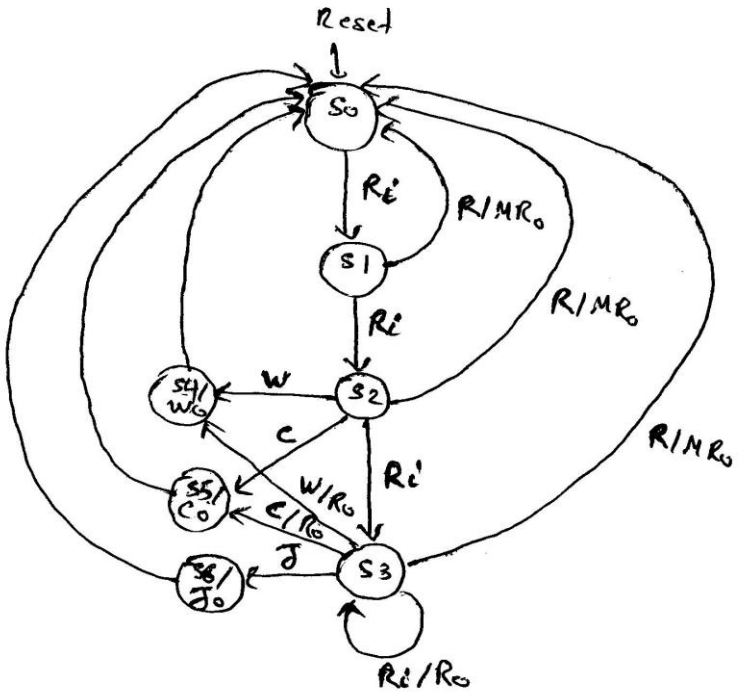


(iii) ASMD Chart:



[13 Points]

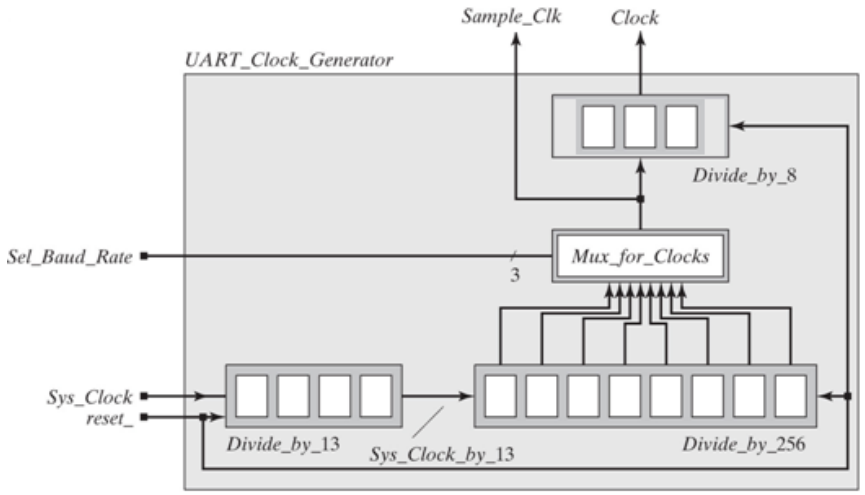
(Q3) It is required to design a **Vending Machine Controller**. The machine takes one Riyal only. For each one Riyal inserted, the input signal **Ri** is asserted. The machine can dispense: Water: 2 Riyals (input **W**), Cola: 2 Riyals (input **C**), Juice: 3 Riyals (input **J**). A product is dispensed in one clock cycle after an input selection. The machine has a money return button (input **R**). When this button is pressed, all the inserted money is returned. It is assumed that only one input may be active at a time. If more than 3 Riyals are inserted, the extra money is automatically returned. If 3 Riyals are inserted and the user selects Water or Cola, the requested product is dispensed along with the return of one Riyal. Outputs are as follows: **MRo**: Money Return Out (all inserted money in the machine), **Wo**: Water Out, **Co**: Cola Out, **Jo**: Juice Out, **Ro**: One Riyal Out. Show a **state diagram** for modeling the Vending Machine Controller.



Note that all unspecified transistions are assumed to keep the state as is.

[13 Points]

(Q4) The functional unit UART_Clock_Generator given below can be used to generate a set of baud rate signal pairs for use in UART.



The table below shows the clock frequency pairs that are generated if Sys_Clock is 8 MHZ.

Sel_Baud_Rate	Sample_Clk (HZ)	Clock (HZ)
000	307,692	38,462
001	153,846	19,231
010	76,923	9,615
011	38,463	4,808
100	19,231	2,404
101	9,615	1,202
110	4,808	601
111	2,404	301

Write a synthesizble Verilog model for the UART_Clock_Generator. The signal **reset_** is an asynchronous active low reset.

module UART_Clock_Generator (output Clock, Sample_Clk, input [2: 0] Sel_Baud_Rate, input Sys_Clock, reset_);

reg [3: 0] temp;
reg [7: 0] temp_by_256;
reg [2: 0] Divide_by_8;

assign Sys_Clock_by_13 = (temp == 4'b1100);
assign Clock = (Divide_by_8== 3'b111);

```
always @(posedge Sys_Clock, negedge reset_)
        if (reset_ == 0) temp <= 0;
        else if (temp == 4'b1100) temp <= 0;
        else temp <= temp + 1;


always @(posedge Sys_Clock_by_13, negedge reset_)
        if (reset_ == 0) temp_by_256 <= 0;
        else temp_by_256 <= temp_by_256 + 1;

always @(posedge Sample_Clk, negedge reset_)
        if (reset_ == 0) Divide_by_8 <= 0;
        else Divide_by_8 <= Divide_by_8 + 1;


assign Sample_Clk = temp_by_256[Sel_Baud_Rate] ; // for the MUX


endmodule
```

[40 Points]

(Q5) Bubble Sort is a well known sorting algorithm. The psuedo code for the sorting algorithm is given below:

```
for i=2 to N
   for j=N downto i
      if [A[j-1] > A[j] then
      begin
         Temp = A[j-1];
         A[j-1] = A[j];
         A[j] = Temp;
      end
```

It is required to write a synthesizable Verilog model for implementing the Bubble Sort algorithm. The ASMD chart and block diagram of a machine that executes the Bubble Sort algorithm is given below.

The machine is parametrizable with the number of words to be sorted as **N** and the word size as **word_size**. It is assumed that once the user sets the **Load** signal to 1, a set of N words will be sent through **Data_in** input serially in the next N clock cycles and loaded into an internal memory. Then when the signal **Sort** is asserted, the algorithm starts sorting. Once sorting is finished and the user asserts **Send** signal, the sorted data will be sent serially through **Data_out** in the next N clock cycles.

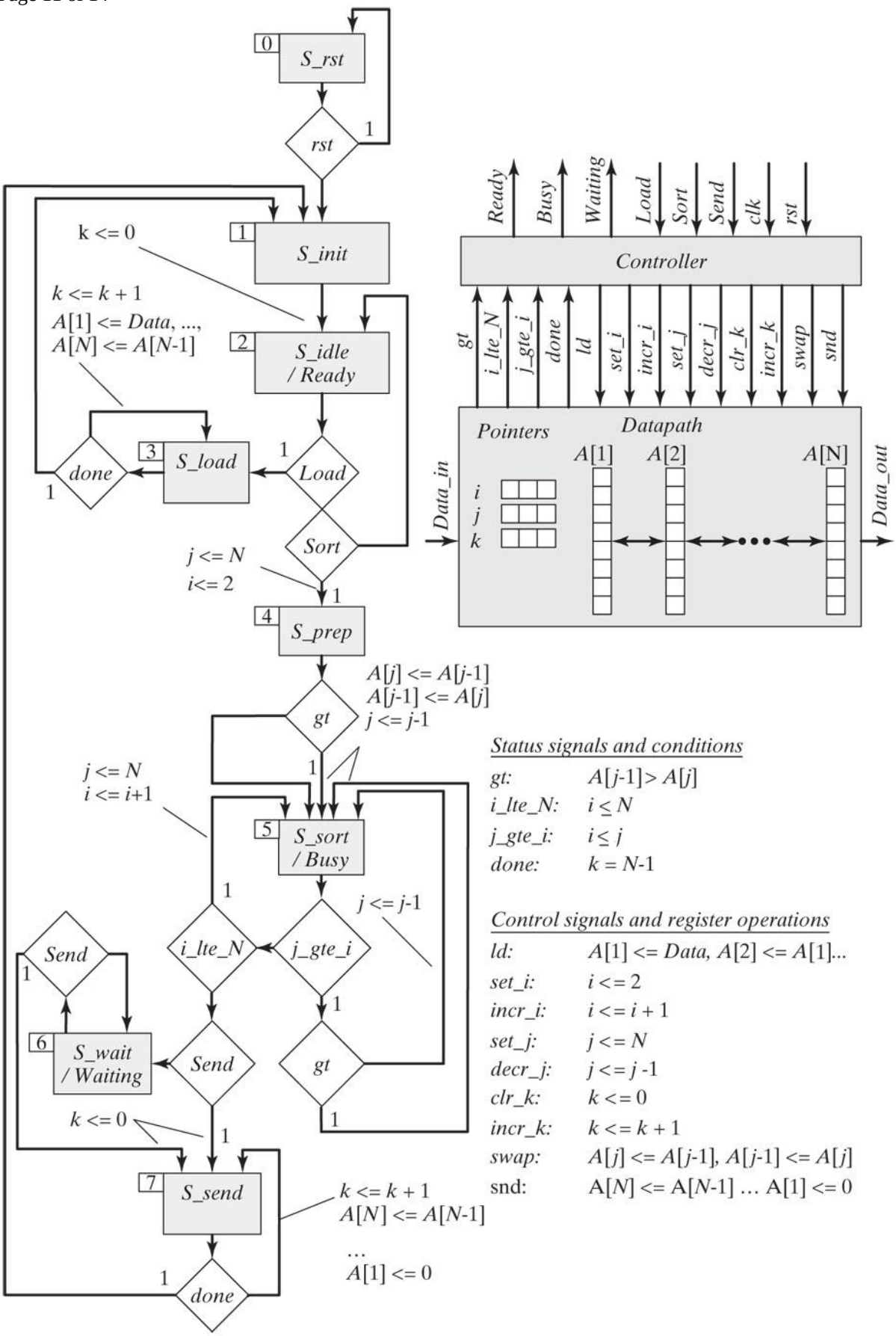
The bubble sort module is given below:

```
module Bubble_Sort # ( parameter N = 8, word_size = 4)(

output [word_size -1: 0] Data_out,
output Ready, Busy, Waiting,
input [word_size -1: 0] Data_in,
input Load, Sort, Send, clk, rst);

Controller M0 (Ready, Busy, Waiting, ld, set_i, incr_i, set_j, decr_j, clr_k, incr_k,
swap, snd, Load, Sort, Send, gt, i_lte_N, j_gte_i, done, clk, rst);

Datapath M1 (Data_out, gt, i_lte_N, j_gte_i, done, Data_in,
ld, set_i, incr_i, set_j, decr_j, clr_k, incr_k, swap, snd, clk);

endmodule
```

(i) Write a parametrizable Verilog model **BSDatapath** to model the datapath of the Bubble_Sort module. The parameters are the number of words to be sorted **N** and the word size **word_size**.

(ii) Write a Verilog model **BSController** to model the control unit of the Bubble_Sort module.

(iii) Write a Verilog **test bench** for the Bubble_Sort module to sort 8 words of test data sent serially as 1 followed by 2, 3, 4, 5, 6, 7 and 8. Assume a CLK of 10 ns period with 50% duty cycle. Your test bench should generate the CLK.



Status signals and conditions

gt:	$A[j-1] > A[j]$
i_lte_N:	$i \leq N$
j_gte_i:	$i \leq j$
done:	$k = N-1$

Control signals and register operations

ld:	$A[1] <= Data$, $A[2] <= A[1]$...
set_i:	$i <= 2$
incr_i:	$i <= i + 1$
set_j:	$j <= N$
decr_j:	$j <= j - 1$
clr_k:	$k <= 0$
incr_k:	$k <= k + 1$
swap:	$A[j] <= A[j-1]$, $A[j-1] <= A[j]$
snd:	$A[N] <= A[N-1] \ldots A[1] <= 0$

(i) DataPath Unit:

```
module BSDatapath # (parameter N = 8, word_size = 4) (
output [word_size -1: 0] Data_out,
output gt, i_lte_N, j_gte_i, done,
input [word_size -1: 0] Data_in,
input ld, set_i, incr_i, set_j, decr_j, clr_k, incr_k, swap, snd, clk);

reg [word_size -1: 0] A [N: 1]; // Array of words
reg [log2(N): 0] i, j;
reg [log2(N)-1: 0] k;
integer ptr;

assign Data_out = A[N];
assign done = (k == N-1);
assign gt = (A[j-1] > A[j]); // compares words
assign i_lte_N = (i <= N);
assign j_gte_i = (i <= j);

always @ (posedge clk) // Datapath and pointers
begin
  if (ld) begin A[1] <= Data_in; for (ptr = 1; ptr < N; ptr = ptr + 1) A[ptr+1] <= A[ptr]; end
  if (swap) begin A[j] <= A[j-1]; A[j-1] <= A[j]; end
  if (decr_j) j <= j-1;
  if (incr_i) i <= i+1;
  if (set_j) j <= N;
  if (set_i) i <= 2;
  if (clr_k) k <= 0;
  if (incr_k) k <= k + 1;
  if (snd) begin A[1] <= 0; for (ptr = 1; ptr < N; ptr = ptr + 1) A[ptr+1] <= A[ptr]; end
end

function integer log2 (input integer n);
    integer i;
    begin
        log2 = 1;
        for (i=0; 2**i<n; i=i+1)
                log2 = i+1;
     end
endfunction

endmodule
```

(ii) Control Unit:

```
module BSController (
output Ready, Busy, Waiting,
output reg ld, set_i, incr_i, set_j, decr_j, clr_k, incr_k, swap, snd,
input Load, Sort, Send, gt, i_lte_N, j_gte_i, done, clk, rst);
```

```
parameter S_rst = 0, S_init = 1, S_idle = 2, S_load = 3, S_prep = 4;
parameter S_sort = 5, S_wait = 6, S_send = 7;
reg [2: 0] state, next_state;

assign Ready = (state == S_idle);
assign Busy = (state == S_sort);
assign Waiting = (state == S_wait);

always @ (posedge clk) if (rst) state <= S_rst; else state <= next_state;

always @ (state, Load, Sort, Send, gt, i_lte_N, j_gte_i, done ) begin
        next_state = S_rst;
        ld = 0;
        set_i = 0; incr_i = 0;
        set_j = 0; decr_j = 0;
        clr_k = 0; incr_k = 0;
        swap = 0; snd = 0;
        case (state)
        S_rst: next_state = S_init;
        S_init:  begin next_state = S_idle; clr_k = 1; end
        S_idle:         if (Load) next_state = S_load;
                                else if (Sort) begin next_state = S_prep; set_j = 1; set_i = 1; end
                                else next_state = S_idle;
        S_load:         if (done) next_state = S_init;
                                else begin next_state = S_load; incr_k = 1; ld = 1; end
        S_prep:         begin next_state = S_sort; if (gt) swap = 1; decr_j = 1; end
        S_sort:         if (j_gte_i) begin next_state = S_sort; decr_j = 1; if (gt) swap = 1; end
                        else if (i_lte_N) begin next_state = S_sort; set_j = 1; incr_i = 1; end
                        else if (Send) begin next_state = S_send; clr_k = 1; end
                        else next_state = S_wait;
        S_wait:         if (Send) begin next_state = S_send; clr_k = 1; end
                                else next_state = S_wait;
        S_send:         if (done) next_state = S_init;
                                else begin next_state = S_send; snd = 1; incr_k = 1; end

        default: next_state = S_rst;
        endcase
end
endmodule
```

(iii) Test Bench:

```
module t_Bubble_Sort ();
parameter word_size = 4;
wire [word_size -1: 0] Data_out;
wire Ready, Busy, Waiting;
reg Load, Sort, Send, clk, rst;
reg [word_size -1: 0] Data_in;
wire [word_size -1: 0] A1, A2, A3, A4, A5, A6, A7, A8;
```

```
assign A1 = M0.M1.A[1];
assign A2 = M0.M1.A[2];
assign A3 = M0.M1.A[3];
assign A4 = M0.M1.A[4];
assign A5 = M0.M1.A[5];
assign A6 = M0.M1.A[6];
assign A7 = M0.M1.A[7];
assign A8 = M0.M1.A[8];
Bubble_Sort M0 (Data_out, Ready, Busy, Waiting, Data_in, Load, Sort, Send, clk,
rst);

initial #700 $finish;
initial begin clk = 0; forever #5 clk = ~clk; end
initial fork
rst = 1; Load = 0; Sort = 0; Send = 0;
#20 rst = 0;
Data_in = 8'h0;
#55 Data_in = 8'h1;
#65 Data_in = 8'h2;
#75 Data_in = 8'h3;
#85 Data_in = 8'h4;
#95 Data_in = 8'h5;
#105 Data_in = 8'h6;
#115 Data_in = 8'h7;
#125 Data_in = 8'h8;
#40 Load = 1;
#50 Load = 0;
#150 Sort = 1;
#160 Sort = 0;
#550 Send = 1;
#560 Send = 0;
join
endmodule
```