Inputs: byte a[256], byte b[256], bit go
Outputs byte a_gt_b, a_eq_b, a_lt_b, bit done

while (1) {
    while (!go);
    done := 0;
    i:=0;
    a_lt_b :=0;
    a_gt_b := 0;
    a_eq_b :=0;
    while ( i  < 256 )) {
        if ( a[i] < b[i] ) a_lt_b := a_lt_b + 1;
        else if ( a[i] == b[i] ) a_eq_b := a_eq_b + 1;
        else a_gt_b := a_gt_b + 1;
        i := i + 1;
    }
    done := 1;
}

1. Convert C code provided above into a high-level finite state machine.
2. Create the datapath for this design, show all connections and components.
3. Connect the datapath to the controller, label all signals.
4. Design the controller FSM. Create a state table showing inputs, current-next state, and outputs.
5. Use binary encoding (e.g. S0 = 000 etc) to encode the states and provide the excitation table.
6. Implement the controller using D-FFs and the excitation table you provided.
7. Show a gate level design of each element in your datapath. Use D-FF where needed.
8. What is the critical path in your design?
   Assume each gate’s delay = 1ns * #gate_inputs. For example, an inverter’s delay is 1ns, while three input NOR gate has 3ns delay. Pass gate has the same delay as an inverter.
9. List elements that are on critical path, and quantify the delay of the critical path.
10. Calculate the maximum clock frequency of your design. For all D-FFs use setup time/hold time of 2ns and clock skew of 3ns. Justify your answer.
11. What dominates the performance and area of your designs?
   Estimate area by counting transistors.