EC-204

<LAB - 9>

NITK SURATHKAL

INBASEKARAN.P

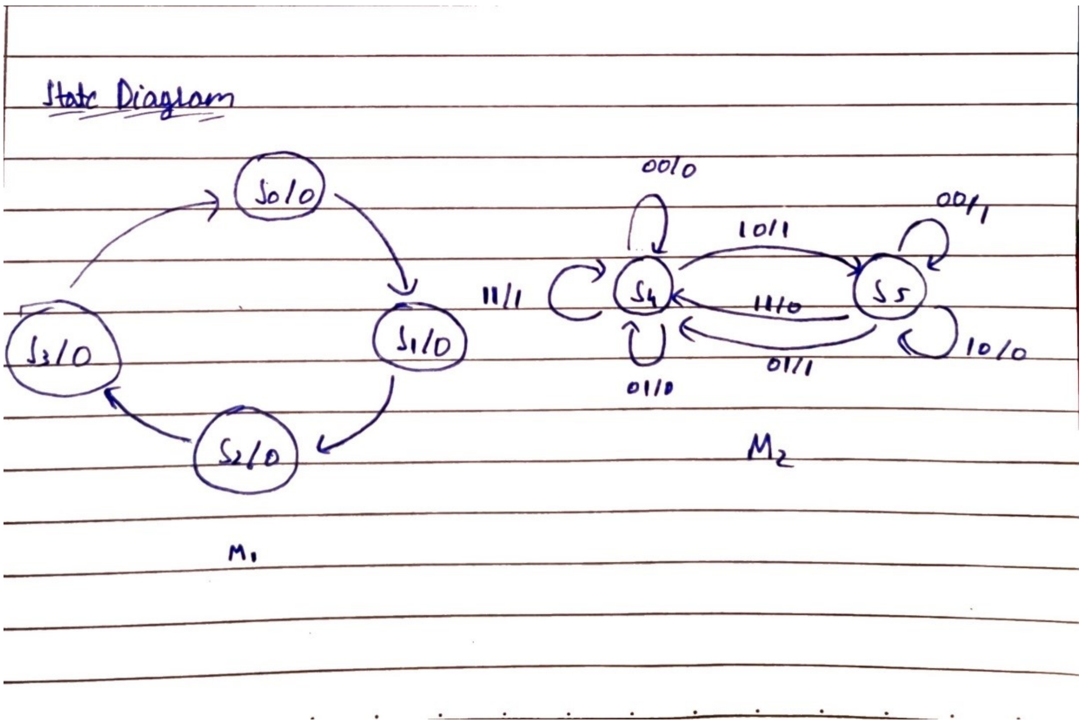
201EC226

Prof: Sumam S

1. A digital system has a single input X and a single output Y

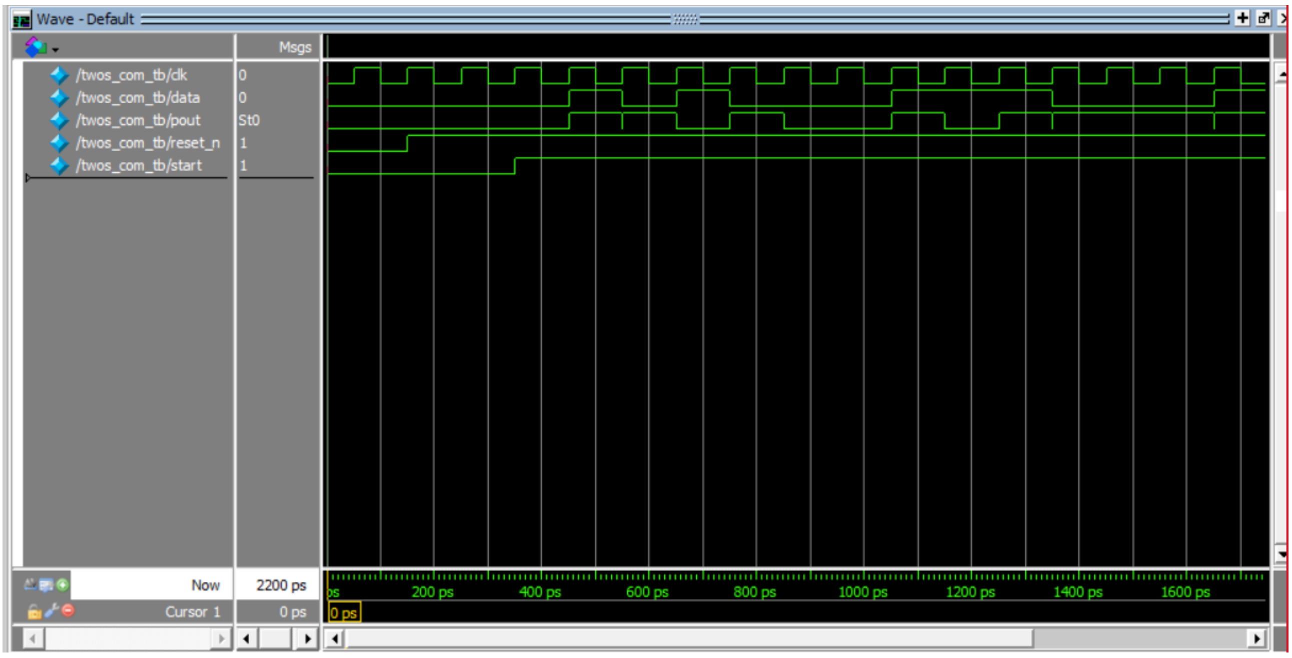
Solution:

State Diagram

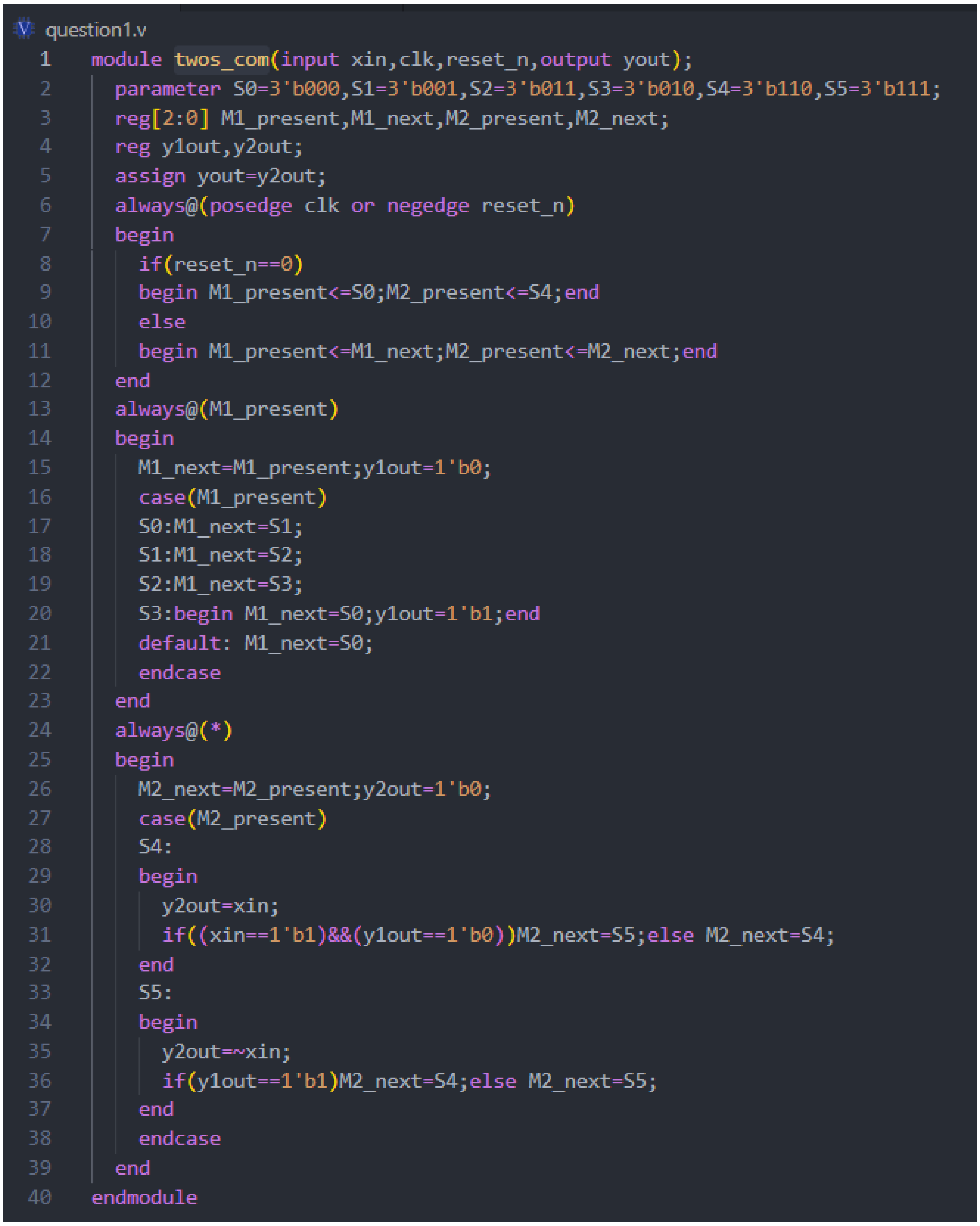


Combining two finite state machines. First one is used to count 4 bits and the second one is used to complement the input stream of bits. S0- S3 belong to M1 and S4-S5 belong to M2.

Simulation



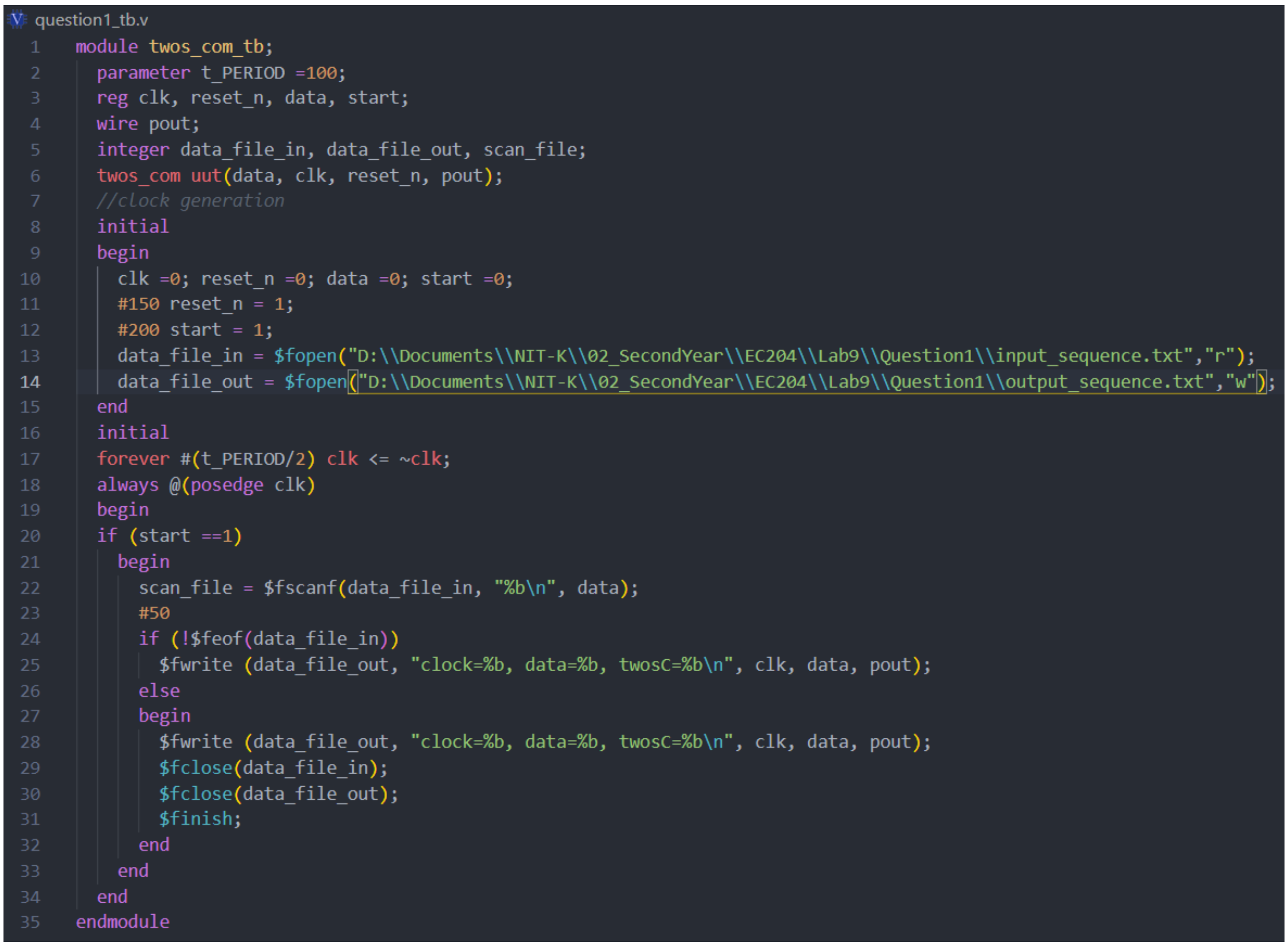
Verilog code



question1.v

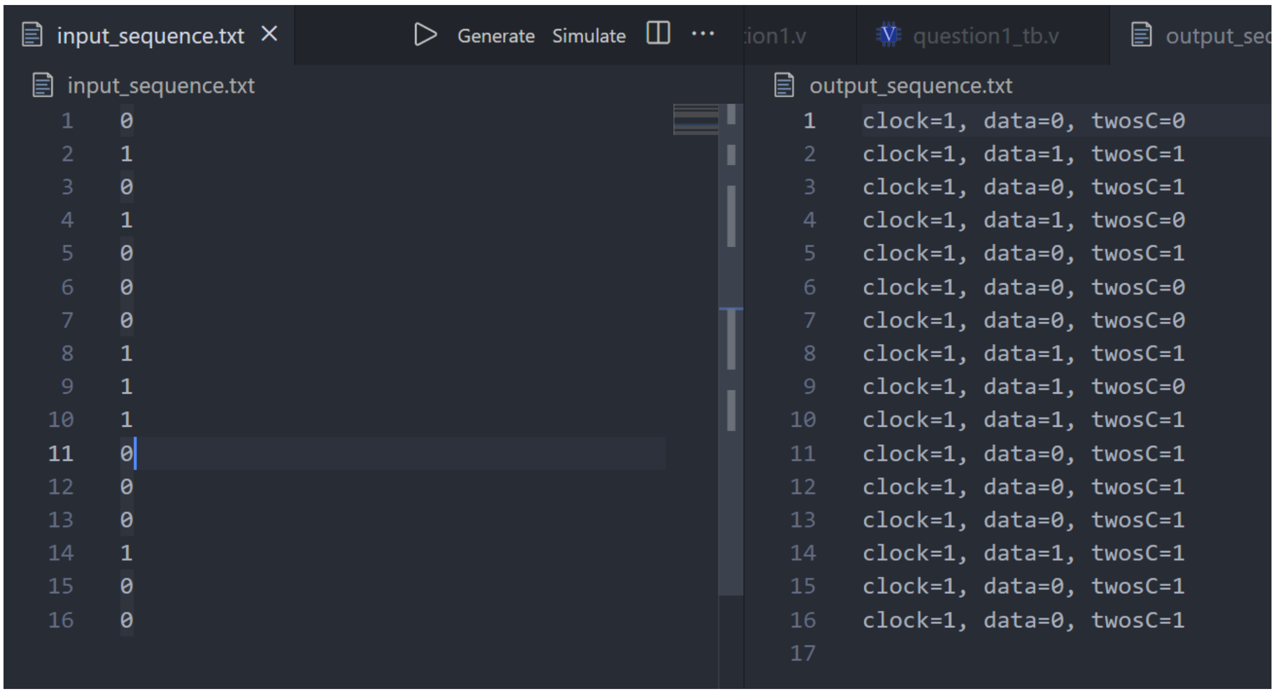
```verilog
module twos_com(input xin,clk,reset_n,output yout);
  parameter S0=3'b000,S1=3'b001,S2=3'b011,S3=3'b010,S4=3'b110,S5=3'b111;
  reg[2:0] M1_present,M1_next,M2_present,M2_next;
  reg y1out,y2out;
  assign yout=y2out;
  always@(posedge clk or negedge reset_n)
  begin
    if(reset_n==0)
    begin M1_present<=S0;M2_present<=S4;end
    else
    begin M1_present<=M1_next;M2_present<=M2_next;end
  end
  always@(M1_present)
  begin
    M1_next=M1_present;y1out=1'b0;
    case(M1_present)
    S0:M1_next=S1;
    S1:M1_next=S2;
    S2:M1_next=S3;
    S3:begin M1_next=S0;y1out=1'b1;end
    default: M1_next=S0;
    endcase
  end
  always@(*)
  begin
    M2_next=M2_present;y2out=1'b0;
    case(M2_present)
    S4:
    begin
      y2out=xin;
      if((xin==1'b1)&&(y1out==1'b0))M2_next=S5;else M2_next=S4;
    end
    S5:
    begin
      y2out=~xin;
      if(y1out==1'b1)M2_next=S4;else M2_next=S5;
    end
    endcase
  end
endmodule
```

Verilog testbench



```verilog
module twos_com_tb;
  parameter t_PERIOD =100;
  reg clk, reset_n, data, start;
  wire pout;
  integer data_file_in, data_file_out, scan_file;
  twos_com uut(data, clk, reset_n, pout);
  //clock generation
  initial
  begin
    clk =0; reset_n =0; data =0; start =0;
    #150 reset_n = 1;
    #200 start = 1;
    data_file_in = $fopen("D:\\Documents\\NIT-K\\02_SecondYear\\EC204\\Lab9\\Question1\\input_sequence.txt","r");
    data_file_out = $fopen("D:\\Documents\\NIT-K\\02_SecondYear\\EC204\\Lab9\\Question1\\output_sequence.txt","w");
  end
  initial
  forever #(t_PERIOD/2) clk <= ~clk;
  always @(posedge clk)
  begin
  if (start ==1)
    begin
      scan_file = $fscanf(data_file_in, "%b\n", data);
      #50
      if (!$feof(data_file_in))
        $fwrite (data_file_out, "clock=%b, data=%b, twosC=%b\n", clk, data, pout);
      else
      begin
        $fwrite (data_file_out, "clock=%b, data=%b, twosC=%b\n", clk, data, pout);
        $fclose(data_file_in);
        $fclose(data_file_out);
        $finish;
      end
    end
  end
endmodule
```

Output



```
0
1
0
1
0
0
0
1
1
1
0
0
0
1
0
0
```

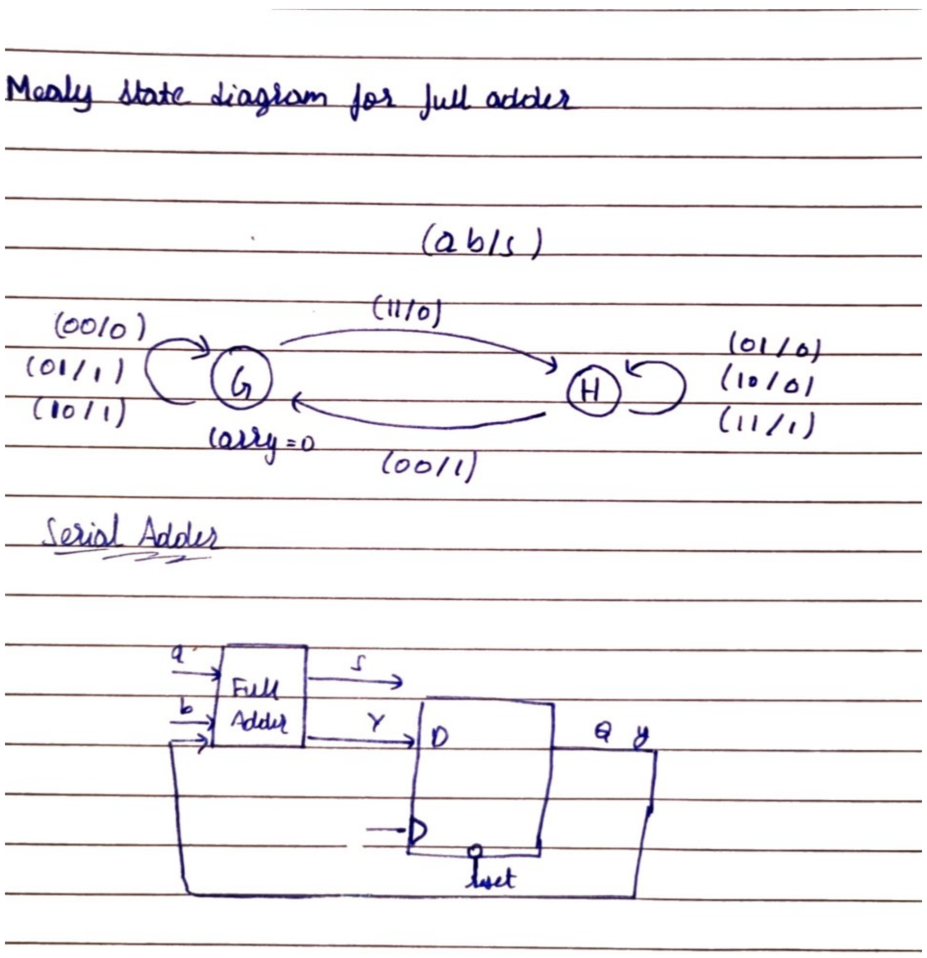
```
clock=1, data=0, twosC=0
clock=1, data=1, twosC=1
clock=1, data=0, twosC=1
clock=1, data=1, twosC=0
clock=1, data=0, twosC=1
clock=1, data=0, twosC=0
clock=1, data=0, twosC=0
clock=1, data=1, twosC=1
clock=1, data=1, twosC=0
clock=1, data=1, twosC=1
clock=1, data=0, twosC=1
clock=1, data=0, twosC=1
clock=1, data=0, twosC=1
clock=1, data=1, twosC=1
clock=1, data=0, twosC=1
clock=1, data=0, twosC=1

```

2. Implement a serial adder in Verilog to add two 8 bit numbers using a single full adder and registers

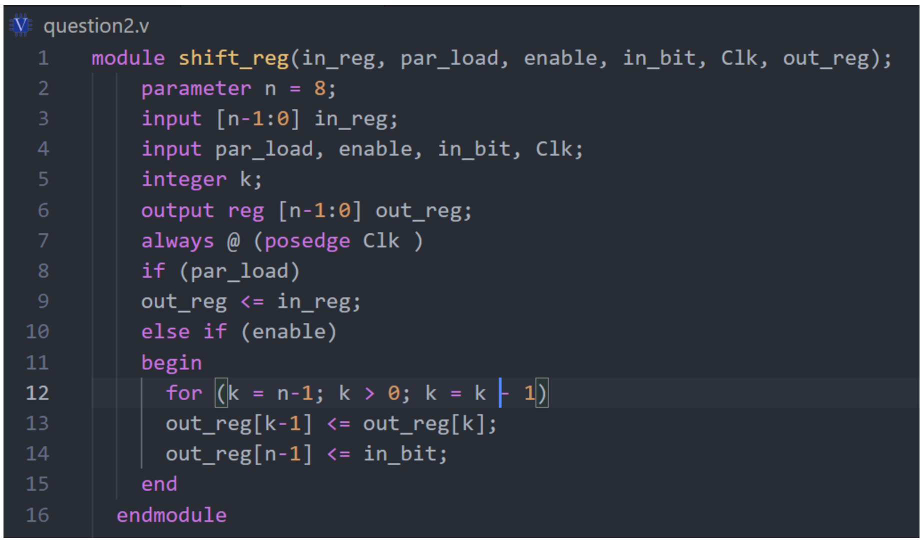
Solution:

State Diagram

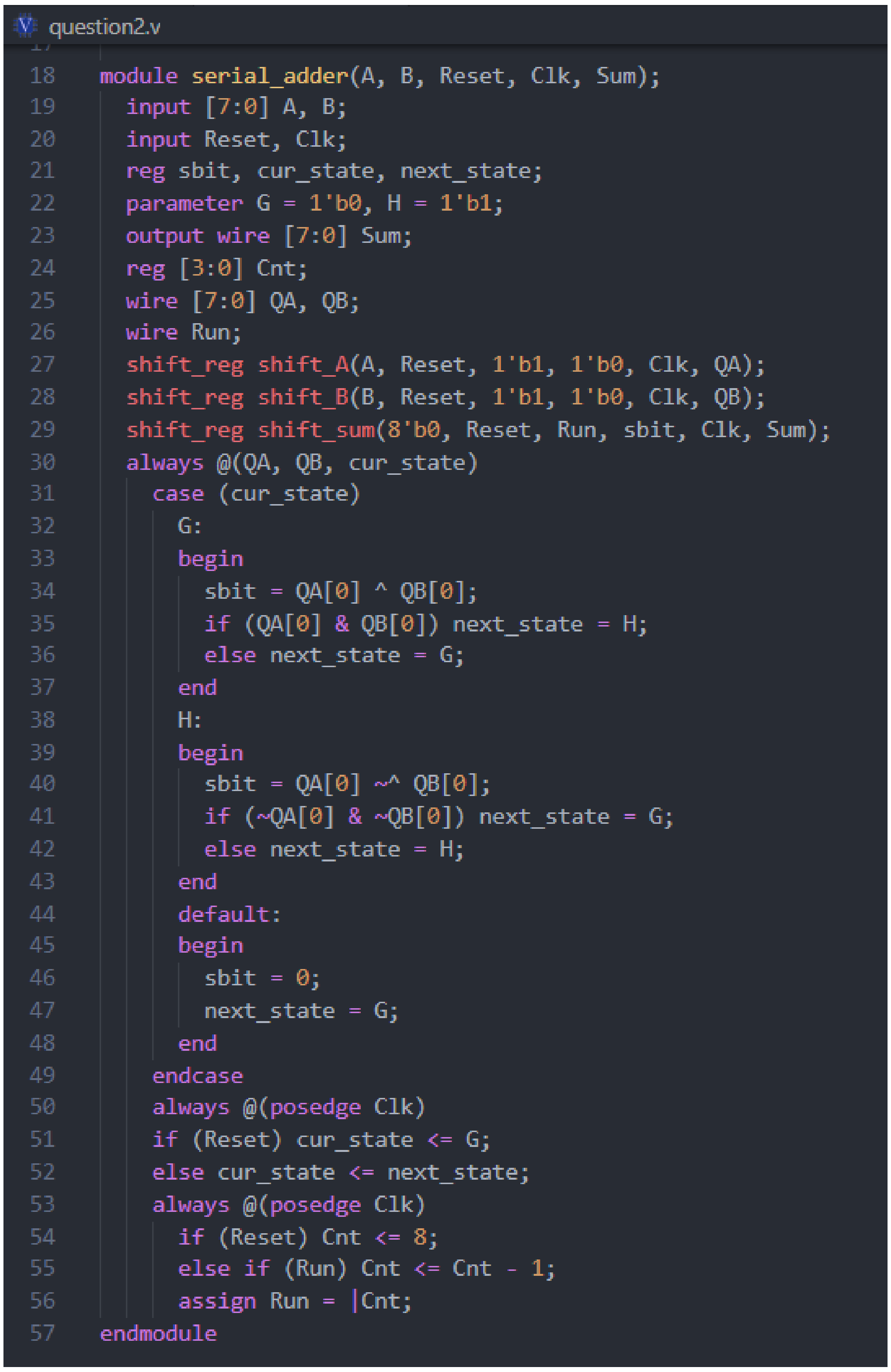


Inputs are two 8 bit numbers. Two shift registers are used to give out the digits which are then serially added. The output is similarly combined using another shift register. A mealy finite state machine is used to implement the full adder and a D flip flop to calculate the carry in for the full adder.

Verilog code


question2.v

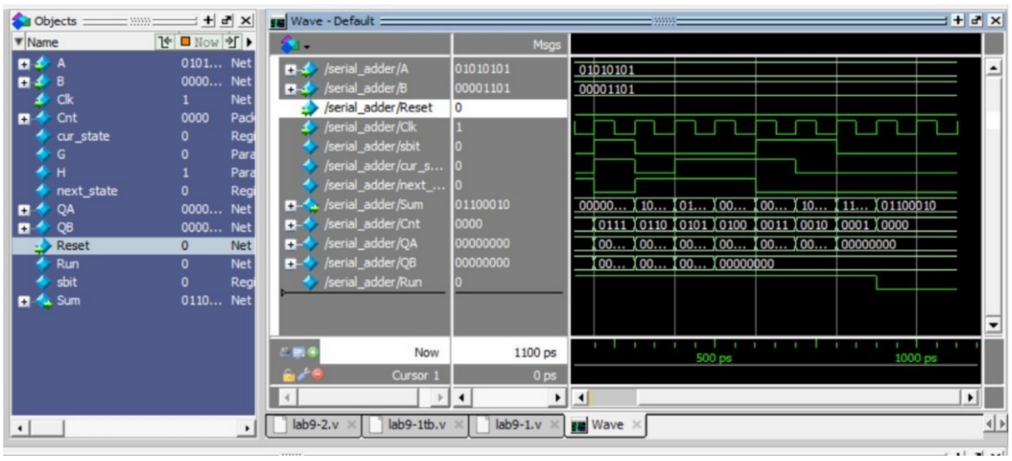
```verilog
module shift_reg(in_reg, par_load, enable, in_bit, Clk, out_reg);
    parameter n = 8;
    input [n-1:0] in_reg;
    input par_load, enable, in_bit, Clk;
    integer k;
    output reg [n-1:0] out_reg;
    always @ (posedge Clk )
    if (par_load)
    out_reg <= in_reg;
    else if (enable)
    begin
      for (k = n-1; k > 0; k = k - 1)
      out_reg[k-1] <= out_reg[k];
      out_reg[n-1] <= in_bit;
    end
  endmodule
```



question2.v

```verilog
module serial_adder(A, B, Reset, Clk, Sum);
  input [7:0] A, B;
  input Reset, Clk;
  reg sbit, cur_state, next_state;
  parameter G = 1'b0, H = 1'b1;
  output wire [7:0] Sum;
  reg [3:0] Cnt;
  wire [7:0] QA, QB;
  wire Run;
  shift_reg shift_A(A, Reset, 1'b1, 1'b0, Clk, QA);
  shift_reg shift_B(B, Reset, 1'b1, 1'b0, Clk, QB);
  shift_reg shift_sum(8'b0, Reset, Run, sbit, Clk, Sum);
  always @(QA, QB, cur_state)
    case (cur_state)
      G:
      begin
        sbit = QA[0] ^ QB[0];
        if (QA[0] & QB[0]) next_state = H;
        else next_state = G;
      end
      H:
      begin
        sbit = QA[0] ~^ QB[0];
        if (~QA[0] & ~QB[0]) next_state = G;
        else next_state = H;
      end
      default:
      begin
        sbit = 0;
        next_state = G;
      end
    endcase
    always @(posedge Clk)
    if (Reset) cur_state <= G;
    else cur_state <= next_state;
    always @(posedge Clk)
      if (Reset) Cnt <= 8;
      else if (Run) Cnt <= Cnt - 1;
      assign Run = |Cnt;
endmodule
```

Simulation



```
Objects
Name          Now
A             0101...   Net
B             0000...   Net
Clk           1         Net
Cnt           0000      Pack
cur_state     0         Reg
G             0         Para
H             1         Para
next_state    0         Reg
QA            0000...   Net
QB            0000...   Net
Reset         0         Net
Run           0         Net
sbit          0         Reg
Sum           0110...   Net
```

```
Wave - Default
                         Msgs
/serial_adder/A          01010101    01010101
/serial_adder/B          00001101    00001101
/serial_adder/Reset      0
/serial_adder/Clk        1
/serial_adder/sbit       0
/serial_adder/cur_s...   0
/serial_adder/next_...   0
/serial_adder/Sum        01100010    00000... 10... 01... 00... 00... 10... 11... 01100010
/serial_adder/Cnt        0000        0111 0110 0101 0100 0011 0010 0001 0000
/serial_adder/QA         00000000    00... 00... 00... 00... 00... 00... 00000000
/serial_adder/QB         00000000    00... 00... 00... 00000000
/serial_adder/Run        0

Now        1100 ps
Cursor 1   0 ps
```