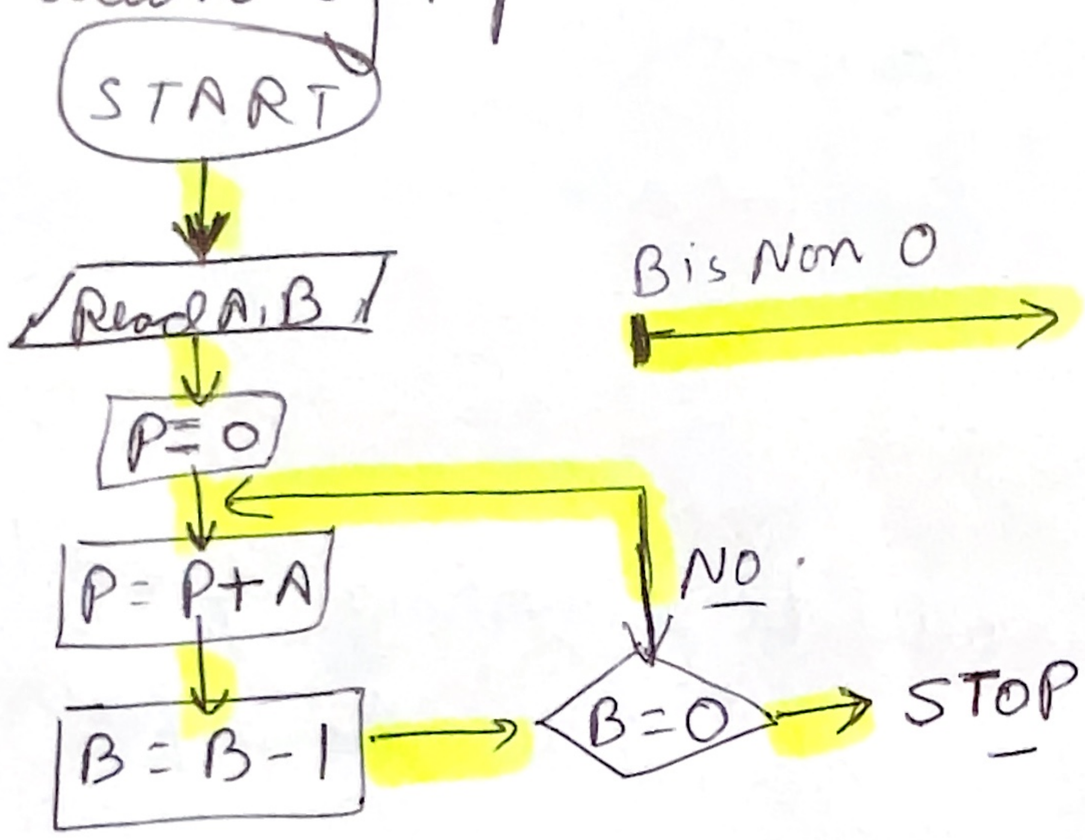
S1 Activating load A [B & C will be stored in A]

S2 Activating load B.

Control Path.

Multiplication by Repeated Addition.



① 3 Reg's.
② Adder
③ Down Counter.
④ Comparator checks for 0.

first datapath.
then FSM & Control Path.

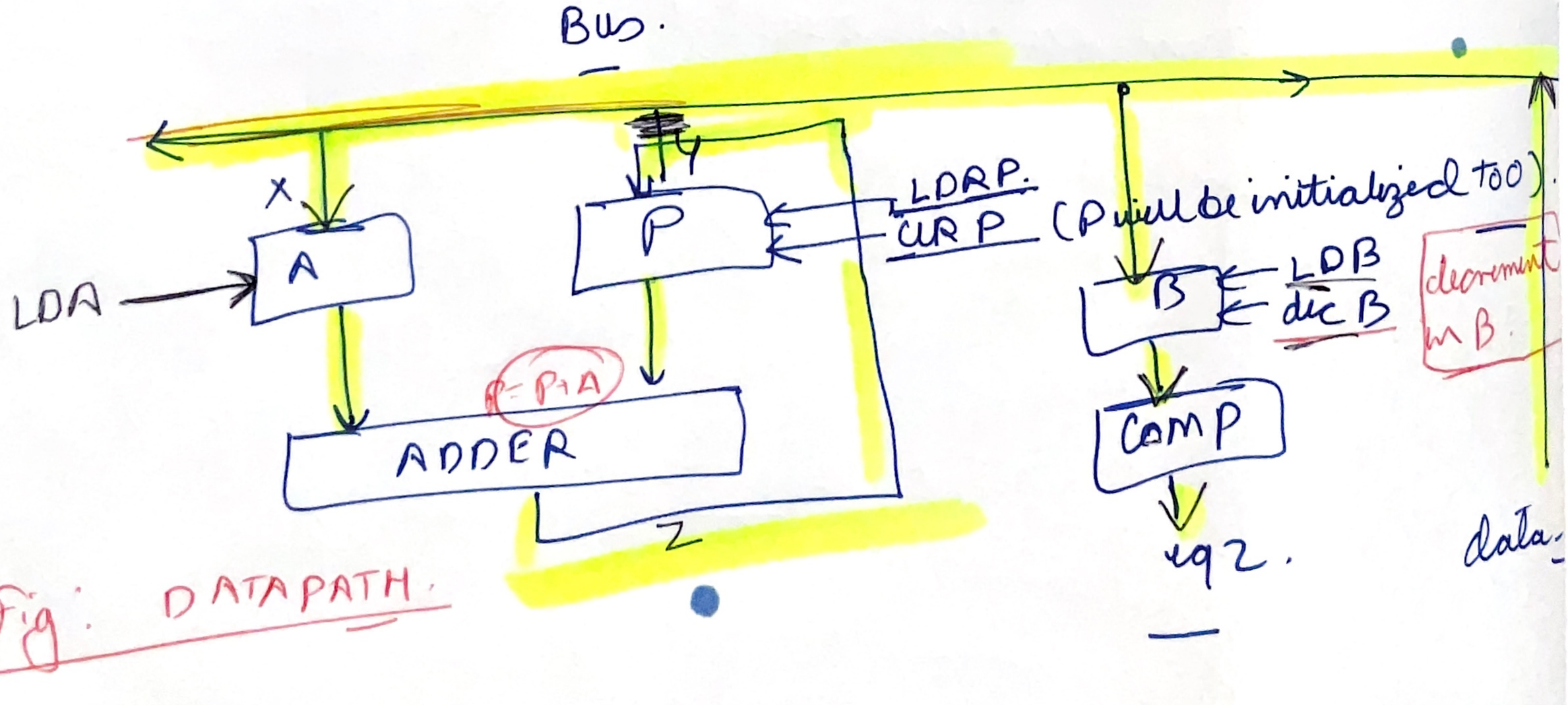
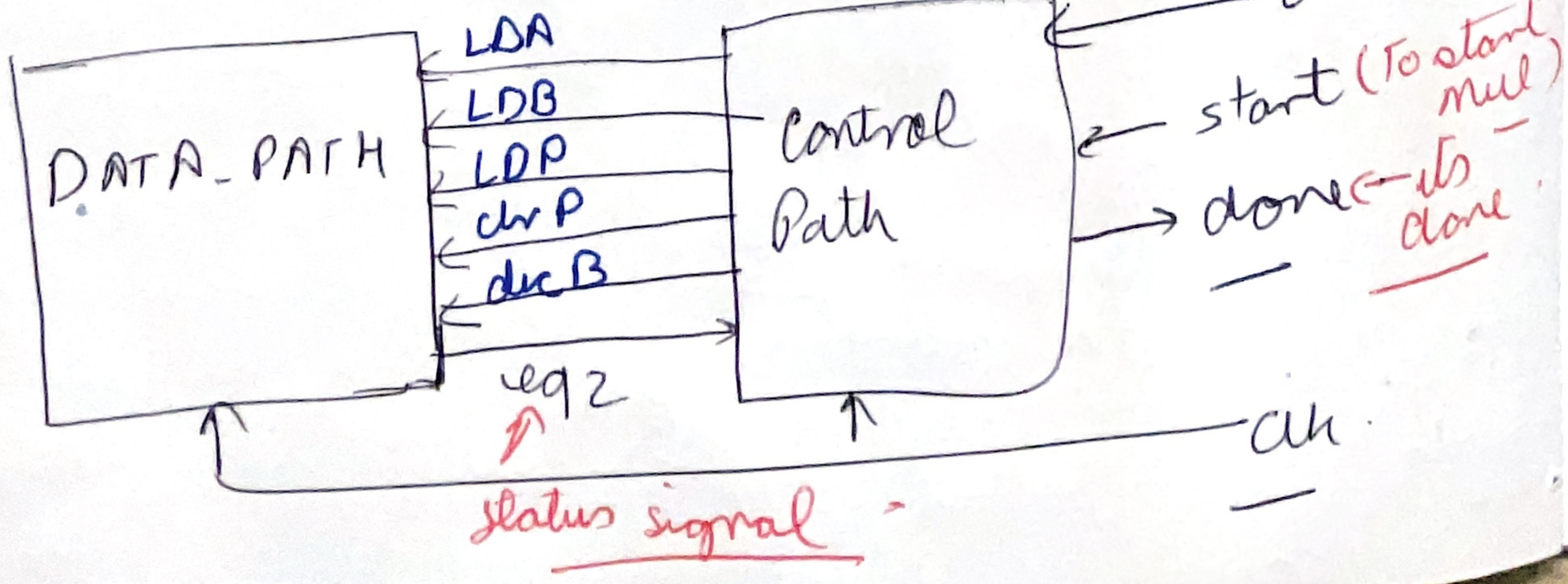
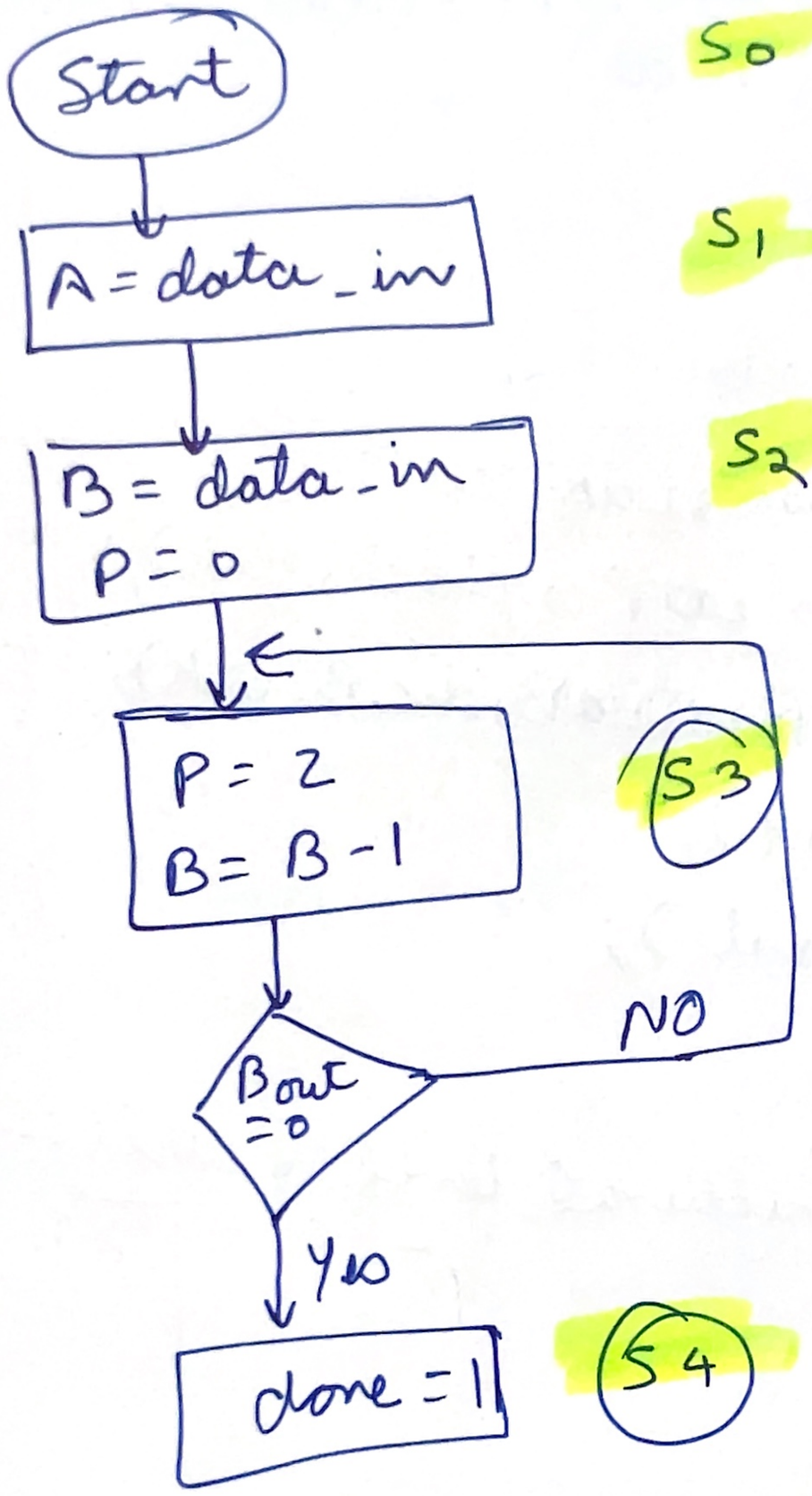


Fig: DATAPATH.

DATA PATH.

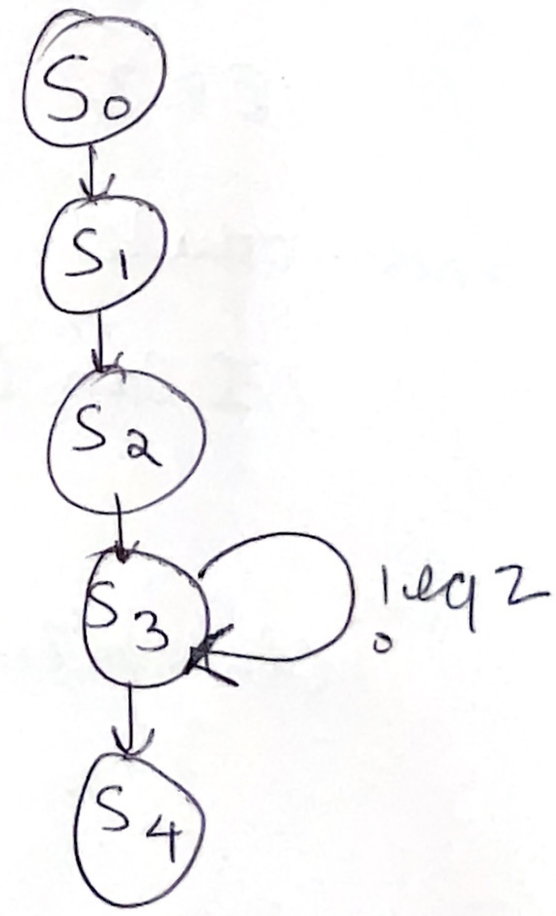


Control Path.



loading in A

loading in B
also
we can
clear P.



```
module mul_datapath (eq2, ldA, LdB, LDP, clrP, dec B, data_in, clk);
    input LDA, LDB, clrP, dec B, clk;
    input [15:0] data_in;
    output eq2;
    wire [15:0] x, y, z, Bout, Bus;

    PIPO1   A   (x, Bus, LdA, clk);        // Reg1
    PIPO2   B   (y, z, LDP, clrP, clk);    // Reg2
    CNTR    B   (Bout, Bus, LdB, dec B, clk);  // Counter
    ADD     AD  (z, x, y);                 // Adder
    EQ2     comp (eq2, Bout);

endmodule.
    // I Can Code them in structural level as well.
```

~~AMAT~~ Observation:

```
module PIPO1 ( dout , din ,ld,clk);
    input wire [15:0] din;
    input ld,clk;
    output reg [15:0] dout;
    always @ (posedge clk)
        if( ld ) dout <= din;

endmodule.
```

// input gets loaded
// if load is active

```
module ADD (out , in1, in2);
    input [15:0] in1,in2;
    output reg [15:0] out;
    always@(*)
        out = in1+in2;

endmodule.
module PIPO2 ( dout , din , ld , clr , clk);
    input [15:0] din;
    input ld , clr , clk;
    output reg [15:0] dout;
    always @ (posedge clk)
        if (clr) dout <= 16'b0;
        else if (ld) dout <= din;

endmodule
module EQZ (eqz, data);
    input [15:0] data;
    output eqz;
    assign eqz = (data == 0);
endmodule.
```

// Can use NOR.

```
module CNTR (dout, din, ld, dec, clk);
    input [15:0] din;
    input ld, dec, clk;
    output reg [15:0] dout;
    always @(posedge clk)
        if(ld) dout <= din;
        else if(dec) dout <= dout - 1;
endmodule
```

// Coding the Controller (FSM).

// ~~IP -> done eq2~~

```
module controller (ldA, ldB, ldP, clrP, decB, done, clk, eq2, start);
    // will be generating the signals
    // LDA, LDB, dPP, dRP, decB, done -- O/P of controller
    // IP -> clk, eq2, start;
    // eq2 Coming from data Path
    // start Coming from outside
    // We have 5 States 0 -> 4
```


```
    input clk, eq2, start;
    output reg ldA, ldB, ldP, clrP, decB, done;
    reg [2:0] state;
    parameter S0 = 3'b000, S1 = 3'b001, S2 = 3'b010,
              S3 = 3'b011, S4 = 3'b100;

    always @(posedge clk)
        begin
            case (state)
                S0 : if(start) state <= S1;
                S1 : state <= S2;
                S2 : state <= S3;
                S3 : #2 if (eq2) state <= S4;   // delay for the purpose of sync
                S4 : state <= S4;
                default : state <= S0;
            endcase
        end
```

// Now To generate signals.

```
always @(state)
  begin
    case (state)
        S0 : begin #1 LDA=0; LDB=0; LDP=0; CLRP=0;
                      decB=0; end
        S1: begin #1 LDA=1; end
        S2: begin #1 LDA=0; LDB=1; CLRP=1; end;
        S3: begin #1 LDB=0; LDP=1; CLRP=0;
                      decB=0; end.
        S4 : begin #1 done=1; LDB=0, LDP=0, decB=0 end
        default: begin #1 LDA=0; LDB=0; LDP=0; CLRP=0;
                      decB=0; end

    endCase
end
endmodule.
```

P=0