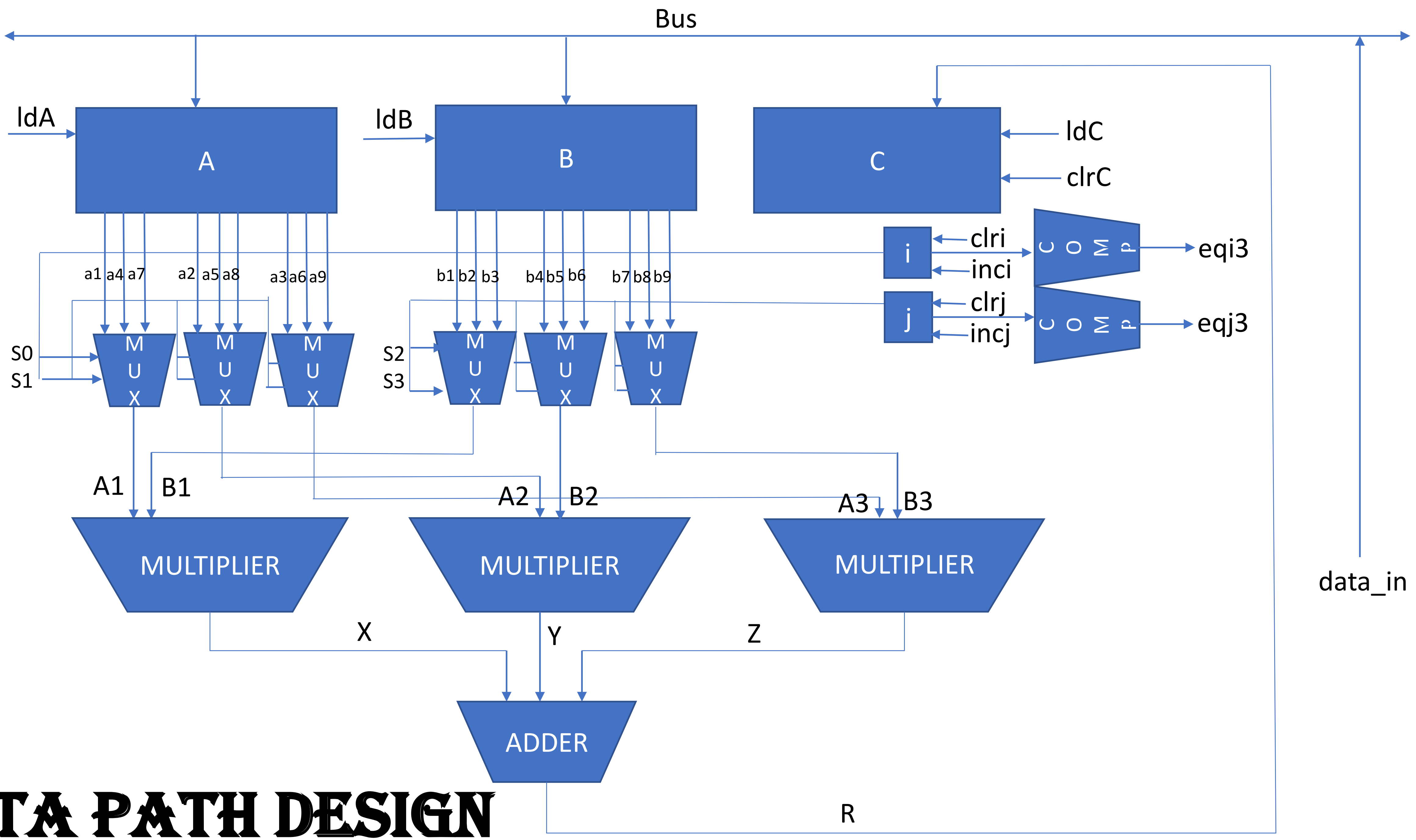
3 X 3 MATRIX MULTIPLIER

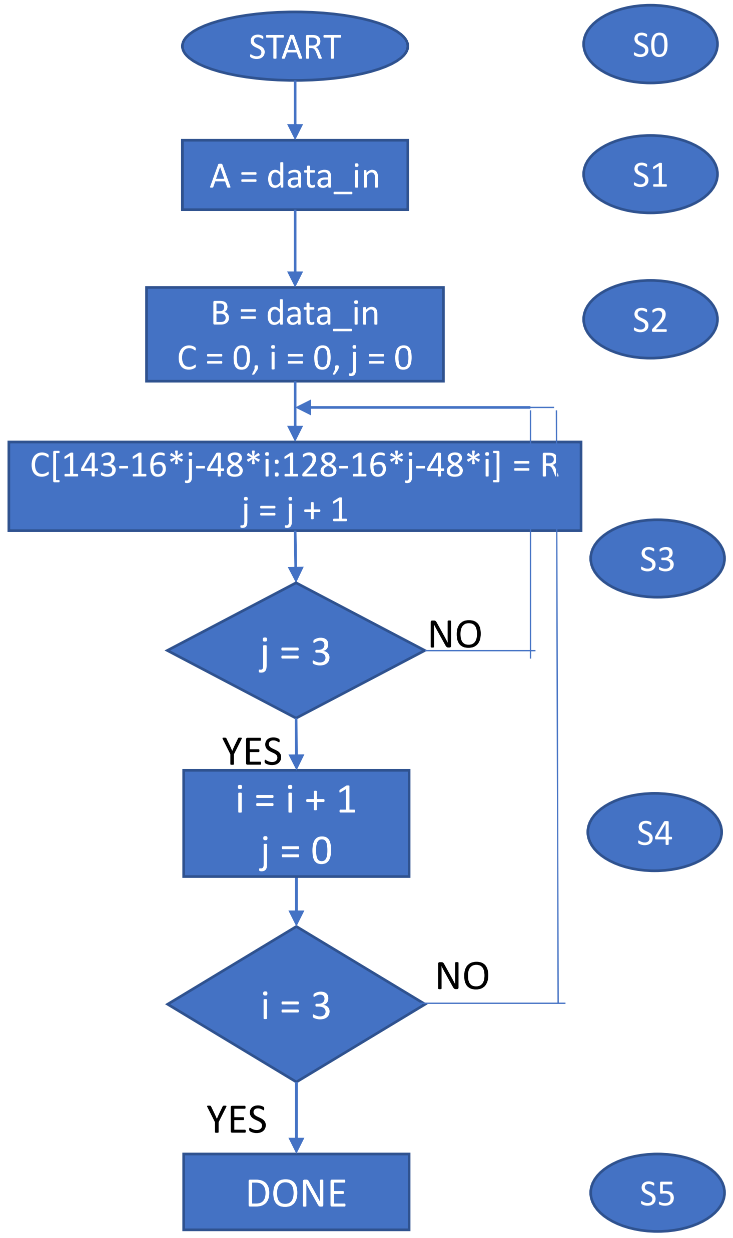
DESCRIPTION:

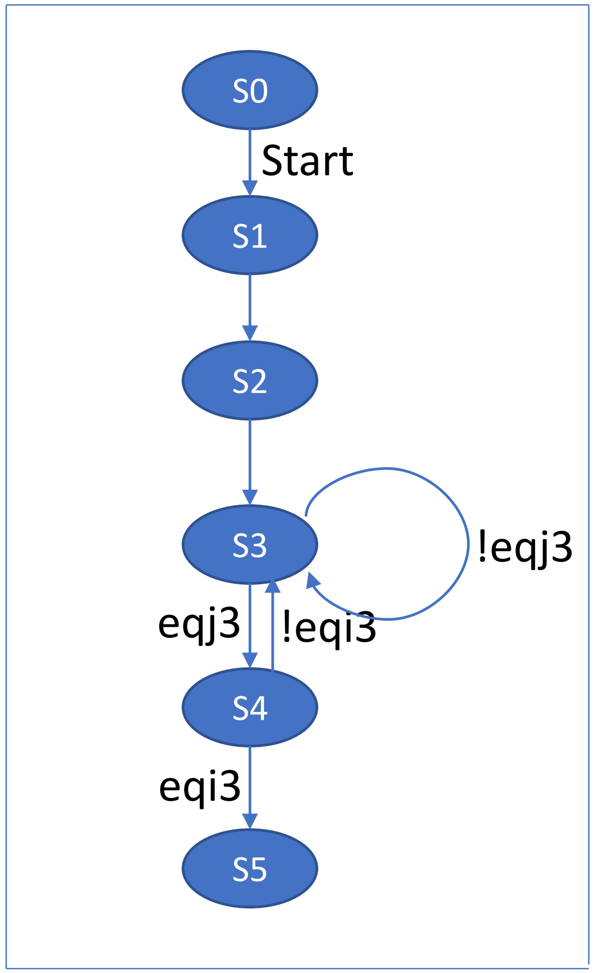
- Implemented the **Datapath and Controller Hardware Design** of **3 x 3 Matrix Multiplier** in **Verilog HDL**.
- The inputs of the design are **two 3 x 3 matrices** consisting of **nine 8-bit numbers** and the processed output is a **3 x 3 matrix** consisting of **nine 16-bit numbers**.
- The design of the Datapath includes **two 72 bit registers** to store the input matrices and **one 144 bit register** to store the output matrix.
- The input data is fed to the hardware through the **bus** in the form of **binary** and is stored in the required registers.
- The inputs are processed accordingly using **three MUXes, three Multipliers, and an Adder** to implement the Matrix Multiplication (using a conventional matrix multiplication algorithm) on the two inputs and the processed output is stored in the required register.
- The Controller design consists an **FSM** of **five states**, So **three flipflops** are needed to implement those five states in FSM.
- A **clock signal** of required frequency is also being fed to the Datapath and Controller.



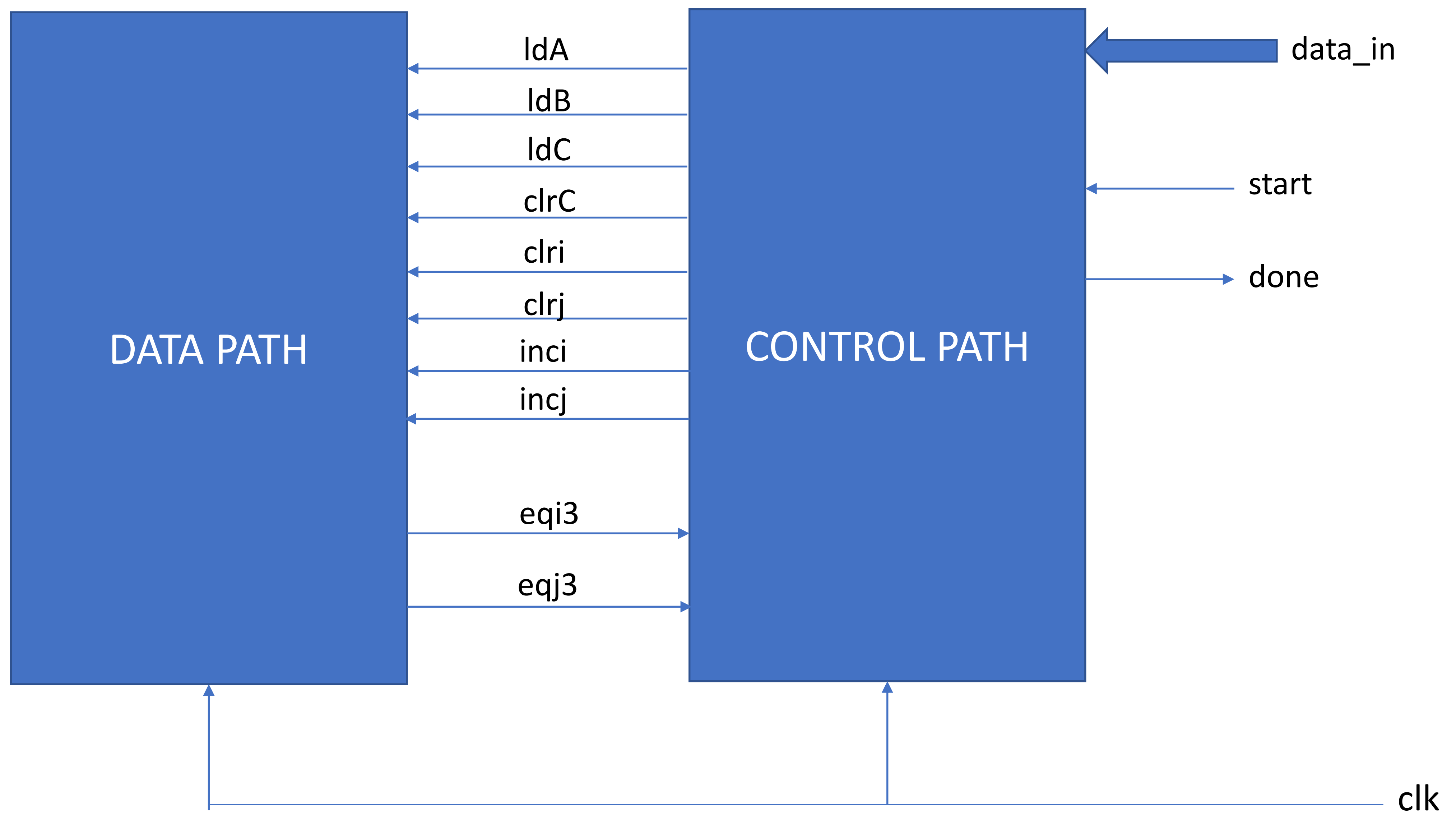
DATA PATH DESIGN

CONTROL PATH DESIGN





COMPLETE DESIGN



VERILOG CODE FOR DATAPATH DESIGN

```
module MatrixMultiplier_datapath (eqi3, eqj3, ldA, ldB, ldC, clri, clrj, inci, incj, data_in, clk);
    input ldA, ldB, ldC, clri, clrj, inci, incj, clk;
    input [71:0] data_in;
    output reg eqi3, eqj3;
    reg [71:0] A, B;
    reg [143:0] C;
    reg [1:0] i, j;
    wire [7:0] A1, A2, A3, B1, B2, B3;
    wire [15:0] X, Y, Z, R

    PIPO1 A (A, data_in, ldA, clk);
    PIPO1 B (B, data_in, ldB, clk);
    PIPO2 C (C, C[143-16*j-48*i:128-16*j-48*i], R, ldC, clk);
    PIPO3 I (i, clri, inci, clk);
    PIPO3 J (j, clrj, incj,  clk);
    MUX MUX_1 (A1, A[71:64], A[57:40], A[23:16], i);
    MUX MUX_2 (A2, A[63:56], A[39:32], A[15:8], i);
    MUX MUX_3 (A3, A[55:48], A[31:24], A[7:0], i);
    MUX MUX_4 (B1, B[71:64], B[63:56], B[55:48], j);
    MUX MUX_5 (B2, B[47:40], B[39:32], B[31:24], j);
    MUX MUX_6 (B3, B[23:16], B[15:8], B[7:0], j);
    MULTIPLY MUL_1 (X, A1, B1):
    MULTIPLY MUL_2 (Y, A2, B2);
    MULTIPLY MUL_3 (Z, A3, B3);
    ADDER ADD (R, X, Y, Z);
    COMPARE COMP (eqi3, i);
    COMPARE COMP (eqj3, j);
endmodule
```

```
module PIPO1 (d_out, d_in, ld, clk);
    input [71:0] d_in;
    input ld, clk;
    output reg [71:0] d_out;
    always @(posedge clk)
            if (ld) dout <= din;
endmodule

module PIPO2 (d_out, d_in, ld, clk);
    input [15:0] d_in;
    input ld, clk;
    output reg [15:0] d_out;
    always @(posedge clk)
            if (ld) d_out  <= d_in;
endmodule

module PIPO3 (d_out, clr, inc, clk);
    input clr, inc, clk;
    output reg [1:0] d_out;
    always @(posedge clk)
            if (clr) d_out <= 2’b0;
            else if (inc) d_out <= d_out + 1;
endmodule
```

```
module MUX (out, in0, in1, in2, sel);
    input [7:0] in0, in1, in2;
    input [1:0] sel;
    output [7:0] out;
    if (sel = 0) assign out = in0;
    if (sel = 1) assign out = in1;
    if (sel = 2) assign out = in2;
endmodule

module MULTIPLY (out, in1, in2);
    input [7:0] in1, in2;
    output [15:0] out;
    always @(*)
                out = in1 * in2;
endmodule

module ADDER (out, in1, in2, in3);
    input [15:0] in1, in2;
    output reg [15:0] out;
    always @(*)
                out = in1 + in2 + in3;
endmodule
```

```
module COMPARE (eq, data);
    input [1:0] data1, data2;
    output eq;
    assign eq = data == 3;
endmodule
```

VERILOG CODE FOR CONTROLLER DESIGN

```
module controller (ldA, ldB, ldC, clri, clrj, inci, incj, done, clk, eqi3, eqj3, start);
    input clk, eqi3, eqj3, start;
    output reg ldA, ldB, ldC, clri, clrj, inci, incj, done;
    reg [3:0] state;
    parameter S0 = 3'b000, S1 = 3'b001, S2 = 3'b010, S3 = 3'b011, S4 = 3'b100, S5 = 3’b101;
    always @(posedge clk)
        begin
            case (state)
                S0 : if (start) state <= S1;
                  S1 : state <= S2;
                  S2 : state <= S3;
                  S3 : #2 if(eqj3) state  <= S4;
                          else state <= S3;
                  S4 : #2 if(eqi3) state  <= S5;
                          else state <= S3;
                  default: state <= S0;
            endcase
        end
```

```
 always @(state)
          begin
               case (state)
                     S0 : begin #1 ldA = 0; ldB = 0; ldC = 0; clri = 0; clrj = 0; inci = 0; incj = 0; end
                          S1 : begin #1 ldA = 1; end
                          S2 : begin #1 ldA = 0; ldB = 1; clri = 1; clrj = 1; end
                          S3 : begin #1 ldB = 0; ldC = 1; incj = 1; clrj = 0; clri =0; end
                          S4 : begin #1 incj = 0; clrj = 1; inci = 1; end
                          S5 : begin #1 done = 1; ldC = 0; inci = 0; clri = 0; clrj = 0; end
                          default: begin #1 1 ldA = 0; ldB = 0; ldC = 0; clri = 0; clrj = 0; inci = 0; incj = 0; end
               endcase
          end
endmodule
```

VERILOG CODE FOR TESTBENCH

```
module MatrixMultiplier_test;
    reg [71:0] data_in;
    reg clk, start, done;
    reg [71:0] A, B;
    reg [143:0] C;
    reg [1:0] i, j;
    MatrixMultiplier_datapath DP (eqi3, eqj3, ldA, ldB, ldC, clrC, clri, clrj, inci, incj, data_in, clk);
    controller CONTROLLER (ldA, ldB, ldC, clrC, clri, clrj, inci, incj, done, clk, eqi3, eqj3, start);
    initial
        begin
                clk = 1'b0;
                #3 start = 1'b1;
                #1000 $finish;
        end
    always #5 clk = ~clk;
    initial
        begin
                #12 data_in = [1:2:3:4:5:6:7:8:9];
                #10 data_in = [2:3:4:5:6:7:8:9:10];
        end
    initial
         begin
                $monitor ($time, " %d %b", DP.C, done);
         end
endmodule
```