

**LAB 0**

**Systolic Array for Applying**

**Matrix Multiplication**

**Submitted to Eng.** **Ahmed Abdelsalam**

**Made by: Ziad Alaa Anis Mohamed Kamal Kassem  
Contact Info:** [**Ziad.Kassem.eng@gmail.com**](mailto:Ziad.Kassem.eng@gmail.com)

[**Linkedin**](https://www.linkedin.com/in/ziad-kassem-887aa1283/)[**Github**](https://github.com/Ziad-1544)

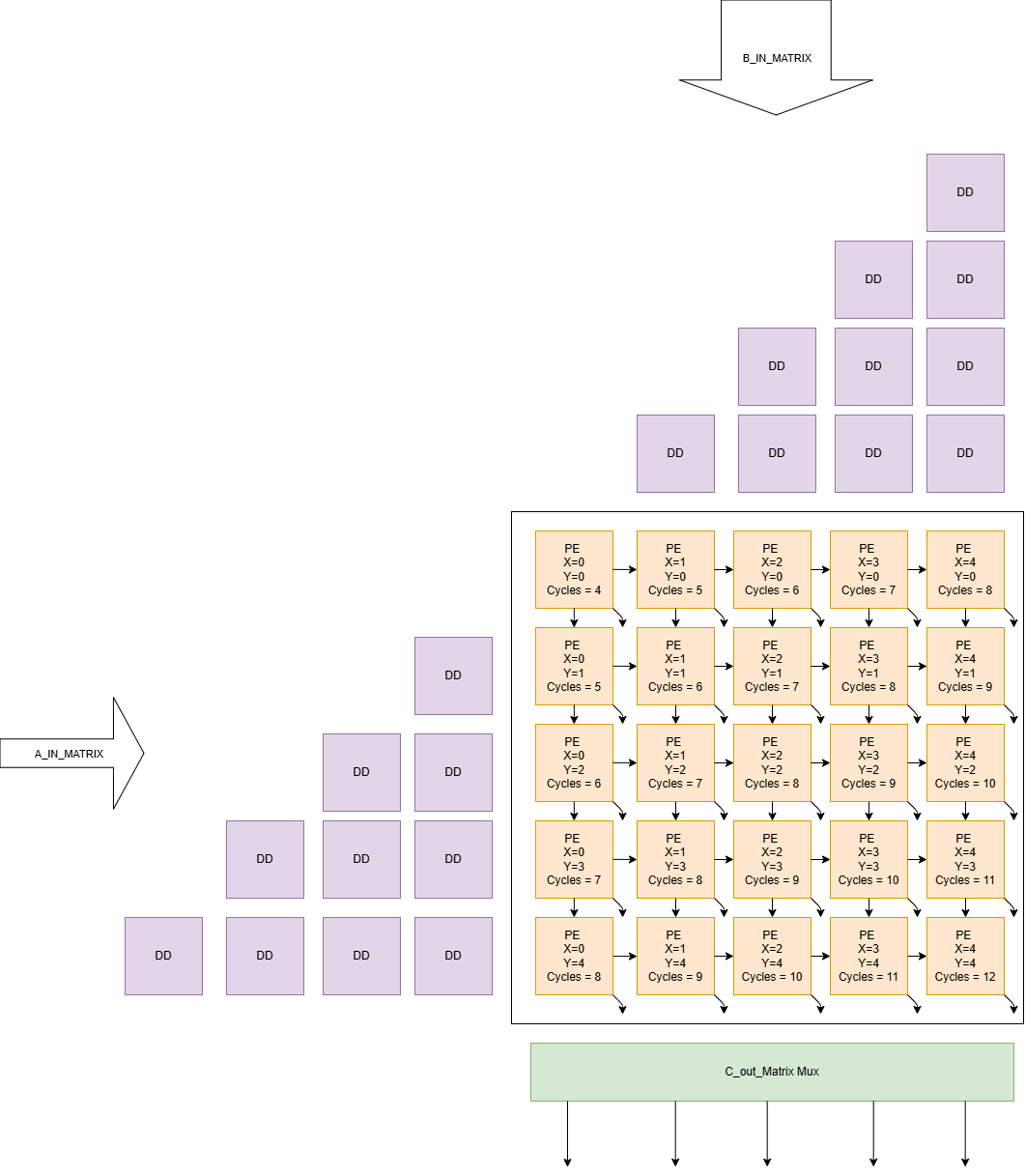
[**Project Repo**](https://github.com/Ziad-1544/Systolic_Array_Paramatrized_Verified)

**Systolic Array Project**

# Intro:

The systolic array is a hardware architecture optimized for matrix multiplication and similar linear algebra computations. It is composed of a two-dimensional grid of simple processing elements (PEs), where each PE performs basic operations such as multiply and accumulate. This structure allows for efficient and parallel data movement and computation, making it ideal for applications like digital signal processing (DSP) and machine learning.

# Design Diagram:



## Data Flow:

**-Matrix A** elements are streamed into the array row-wise from the **left**, and  
**-Matrix B** elements are streamed column-wise from the **top**.

Each PE:

* Receives one element from Matrix A and one from Matrix B.
* Multiplies the two input values.
* Adds the result to its internal accumulator register.
* Passes:
  + The A value to the right neighbor.
  + The B value to the bottom neighbor.

## Computation Steps

At each clock cycle:

1. The first elements of A and B enter the top-left PE (position (0,0)).
2. The PE multiplies these values and stores the result.
3. On subsequent cycles, new A and B elements enter from the left and top edges respectively.
4. Each PE performs a multiply-accumulate operation and shifts data to its neighbors.
5. After (N\_SIZE + X\_id +Y\_id) cycles, each PE will have computed the full value of its corresponding.

Each Processing Element (PE) in the systolic array includes an internal counter that plays a critical role in synchronization and control. The counter is responsible for tracking the computation cycles within the PE. Since matrix multiplication requires summing over (2 \* N\_SIZE) elements (i.e., performing (N\_SIZE) multiply-accumulate operations per PE), the counter keeps track of how many valid data items have passed through the PE. This ensures that the accumulation is performed exactly (N\_SIZE) times before producing the final result. The counter typically increments on every clock cycle when valid data is present. Once the counter reaches the value (N\_SIZE + X\_id +Y\_id), it signals that the PE has completed its contribution to the output matrix. At this point, the PE can assert a valid\_out signal, indicating that the result stored in its accumulator is ready to be read. Additionally, this counter-based logic helps in stalling or disabling unnecessary operations after the computation is done, optimizing power and performance in hardware implementations.

**Benefits of the Systolic Array**

* **Parallelism:** All PEs work concurrently, enabling high throughput.
* **Data Locality:** No need to store full matrices in memory.
* **Scalability:** Easily expandable for larger matrix sizes.
* **Efficiency:** Suitable for hardware implementations with regular, predictable data flow.

# Design Code:

module systolic\_array #(

    parameter N\_SIZE = 5 ,

    parameter DATAWIDTH = 16

) (

    input logic clk,rst\_n,valid\_in, *// Asynchronous Active low*

    input wire signed [DATAWIDTH-1:0] matrix\_a\_in [0:N\_SIZE-1], *//Data Array  [COLOMN][DATA]*

    input wire signed [DATAWIDTH-1:0] matrix\_b\_in [0:N\_SIZE-1], *//Data Array  [COLOMN][DATA]*

    output logic valid\_out,

    output logic signed [2\*DATAWIDTH-1:0]matrix\_c\_out[0:N\_SIZE-1]

*// Data Array [Each block in last row OUTPUT][DATA]*

);

*//Local Variables:*

logic [DATAWIDTH-1:0] matrix\_a\_delayed [0:N\_SIZE-1][0:N\_SIZE-1]; *// [ROW][STAGE][DATA]*

logic [DATAWIDTH-1:0] matrix\_b\_delayed [0:N\_SIZE-1][0:N\_SIZE-1]; *// [COLOMN][STAGE][DATA]*

logic [DATAWIDTH-1:0] a\_interconnects [0:N\_SIZE-1][0:N\_SIZE-1]; *// a\_in connections [ROW][COLOMN][DATA]*

logic [DATAWIDTH-1:0] b\_interconnects [0:N\_SIZE-1][0:N\_SIZE-1]; *// b\_in connections [COLOMN][ROW][DATA]*

logic valid\_out\_array [N\_SIZE-1:0][N\_SIZE-1:0];

logic signed [2\*DATAWIDTH-1:0] row\_c\_out [0:N\_SIZE-1][0:N\_SIZE-1];

*//Generates iterators:*

genvar row\_h, col\_h;

genvar row\_v, col\_v;

*//Instantiating Horizontal Flip-Flops:*

generate

    for (row\_h = 0; row\_h < N\_SIZE; row\_h++) begin : GEN\_ROW\_H

        for (col\_h = 0; col\_h <= row\_h; col\_h++) begin : GEN\_COL\_H\_A

            if (col\_h == 0) begin

                assign matrix\_a\_delayed[row\_h][col\_h] = matrix\_a\_in[row\_h];

            end else begin

                D\_ff #(.*DATAWIDTH*(DATAWIDTH)) dff\_a (

                    .*clk*(clk),

                    .*rst\_n*(rst\_n),

                    .*d\_in*(matrix\_a\_delayed[row\_h][col\_h-1]),

                    .*d\_out*(matrix\_a\_delayed[row\_h][col\_h])

                );

            end

        end

    end

endgenerate

*//Instantiating Vertical Flip-Flops:*

generate

    for (col\_v = 0; col\_v<N\_SIZE ; col\_v++ ) begin : GEN\_COL\_v

        for (row\_v =0; row\_v<=col\_v ;row\_v++ ) begin : GEN\_ROW\_V\_B

            if (row\_v == 0) begin

                assign matrix\_b\_delayed[col\_v][row\_v] = matrix\_b\_in[col\_v];

            end else begin

                D\_ff #(.*DATAWIDTH*(DATAWIDTH)) dff\_b (

                    .*clk*(clk),

                    .*rst\_n*(rst\_n),

                    .*d\_in*(matrix\_b\_delayed[col\_v][row\_v-1]),

                    .*d\_out*(matrix\_b\_delayed[col\_v][row\_v]));

            end

        end

    end

endgenerate

*//Connecting interconnected to delayed matrix:*

generate

    for (genvar row = 0 ; row<N\_SIZE ;row++ ) begin :GEN\_ROW\_ASSIGN

        assign a\_interconnects[row][0] = matrix\_a\_delayed[row][row];

    end

    for (genvar col = 0 ; col<N\_SIZE ;col++ ) begin :GEN\_COL\_ASSIGN

        assign b\_interconnects[col][0] = matrix\_b\_delayed[col][col];

    end

endgenerate

*//PE Instantiation:*

genvar row\_PE, col\_PE;

generate

    for (row\_PE = 0; row\_PE<N\_SIZE ; row\_PE++ ) begin :GEN\_PE\_R

        for (col\_PE = 0; col\_PE<N\_SIZE ; col\_PE++ ) begin :GEN\_PE\_C

                PE #(

                .*N\_SIZE*(N\_SIZE),

                .*DATAWIDTH*(DATAWIDTH),

                .*x\_id*(row\_PE),

                .*y\_id*(col\_PE)

                ) pe\_inst (

                .*clk*(clk),

                .*rst\_n*(rst\_n),

                .*valid\_in*(valid\_in),

                .*a\_in*(a\_interconnects[row\_PE][col\_PE]),

                .*b\_in*(b\_interconnects[col\_PE][row\_PE]),

                .*valid\_out*(valid\_out\_array[row\_PE][col\_PE]),

                .*a\_out\_right*(a\_interconnects[row\_PE][col\_PE+1]),

                .*b\_out\_down*(b\_interconnects[col\_PE][row\_PE+1]),

                .*c\_out*(row\_c\_out[row\_PE][col\_PE]));

       end

    end

endgenerate

*// valid\_out logic: high when any row's last PE is valid*

always\_comb begin

    valid\_out = 0;

    for (int row = 0; row < N\_SIZE; row++) begin

        if (valid\_out\_array[row][N\_SIZE-1]) begin

            valid\_out = 1;

        end

    end

end

*//OUTPUT MUX*

always\_comb begin

    matrix\_c\_out = '{default:0};

    for (int row = 0; row < N\_SIZE; row++) begin

        if (valid\_out\_array[row][N\_SIZE-1]) begin

            for (int col = 0; col < N\_SIZE; col++) begin

                matrix\_c\_out[col] = row\_c\_out[row][col];

            end

        end

    end

end

endmodule

module PE #(

    parameter N\_SIZE = 5,

    parameter DATAWIDTH = 16,

    parameter x\_id = 0 ,

    parameter y\_id = 0

) (

    input logic clk,rst\_n,valid\_in,

    input logic signed [DATAWIDTH-1:0] a\_in,b\_in,

    output logic valid\_out,

    output logic signed [(DATAWIDTH)-1:0]b\_out\_down,

    output logic signed [(DATAWIDTH)-1:0]a\_out\_right,

    output logic signed [(2\*DATAWIDTH)-1:0]c\_out

);

logic [N\_SIZE:0] internal\_counter;

logic [(2\*DATAWIDTH)-1:0] sum;

logic valid\_in\_flag\_holder;

    always @(posedge clk , negedge rst\_n)begin

       if (!rst\_n) begin

            internal\_counter <= 0;

            sum <= 0;

            valid\_in\_flag\_holder <=0;

            c\_out<=0;

       end

       else begin

        valid\_out<=0;

        if (valid\_in || valid\_in\_flag\_holder) begin

            valid\_in\_flag\_holder<=1;

            if (internal\_counter < N\_SIZE + x\_id + y\_id) begin

                internal\_counter <= internal\_counter + 1;

                if (internal\_counter >= x\_id +y\_id) begin

                    if (internal\_counter == N\_SIZE + x\_id + y\_id - 1) begin

                        valid\_out <= 1;

                        valid\_in\_flag\_holder <= 0;    internal\_counter <= 0;

                        c\_out <= sum + (a\_in\*b\_in);

                        sum <= 0;

                    end

                    else begin

                        sum <= sum + (a\_in\*b\_in);

                    end

                    a\_out\_right <= a\_in;   b\_out\_down <= b\_in;

                end

            end

            else begin

            end

        end

       end

    end

endmodule

module D\_ff #(

    parameter DATAWIDTH = 16

) (

    input logic clk,rst\_n,

    input logic [DATAWIDTH-1:0] d\_in,

    output logic [DATAWIDTH-1:0] d\_out

);

always @(posedge clk , negedge rst\_n) begin

    if(!rst\_n) d\_out <= 0;

    else d\_out <= d\_in;

end

endmodule

# Verification Code:

module systolic\_array\_tb;

    localparam N\_SIZE = 5;

    localparam DATAWIDTH = 16;

    logic clk, rst\_n, valid\_in;

    logic signed [DATAWIDTH-1:0] matrix\_a\_in [N\_SIZE-1:0];

    logic signed [DATAWIDTH-1:0] matrix\_b\_in [N\_SIZE-1:0];

    logic valid\_out;

    logic signed [2\*DATAWIDTH-1:0] matrix\_c\_out [N\_SIZE-1:0];

*// Instantiate DUT*

    systolic\_array #(

        .*N\_SIZE*(N\_SIZE),

        .*DATAWIDTH*(DATAWIDTH)

    ) dut (

        .*clk*(clk),

        .*rst\_n*(rst\_n),

        .*valid\_in*(valid\_in),

        .*matrix\_a\_in*(matrix\_a\_in),

        .*matrix\_b\_in*(matrix\_b\_in),

        .*valid\_out*(valid\_out),

        .*matrix\_c\_out*(matrix\_c\_out)

    );

*// Clock generation*

    initial begin

        clk=0;

        forever begin

            #1 clk=~clk;

        end

    end

*// Test vectors*

    initial begin

        rst\_n = 0;

        valid\_in = 0;

        matrix\_a\_in = '{default:0};

        matrix\_b\_in = '{default:0};

*repeat*(1) @(negedge clk);

        rst\_n = 1;

*// Test Case 1 5\*5*

        matrix\_a\_in = '{1, 2, 3, 4, 5};

        matrix\_b\_in = '{25, 24, 23, 22, 21};

        valid\_in = 1;

*repeat*(1) @(negedge clk);

        matrix\_a\_in = '{6, 7, 8, 9, 10};

        matrix\_b\_in = '{20, 19, 18, 17, 16};

*repeat*(1) @(negedge clk);

        matrix\_a\_in = '{11, 12, 13, 14, 15};

        matrix\_b\_in = '{15, 14, 13, 12, 11};

*repeat*(1) @(negedge clk);

        matrix\_a\_in = '{16, 17, 18, 19, 20};

        matrix\_b\_in = '{10, 9, 8, 7, 6};

*repeat*(1) @(negedge clk);

        matrix\_a\_in = '{21, 22, 23, 24, 25};

        matrix\_b\_in = '{5, 4, 3, 2, 1};

*repeat*(1) @(negedge clk);

        valid\_in = 0;

*repeat*(13) @(negedge clk);

*$display*("Test 1 should be done");

*$stop*;

*//=================================================================*

*// Test Case 2 5\*5*

        matrix\_a\_in = '{5, 4, 3, 2, 1};

        matrix\_b\_in = '{1, 1, 1, 1, 1};

        valid\_in = 1;

*repeat*(1) @(negedge clk);

        matrix\_a\_in = '{10, 9, 8, 7, 6};

        matrix\_b\_in = '{2, 2, 2, 2, 2};

*repeat*(1) @(negedge clk);

        matrix\_a\_in = '{15, 14, 13, 12, 11};

        matrix\_b\_in = '{3, 3, 3, 3, 3};

*repeat*(1) @(negedge clk);

        matrix\_a\_in = '{20, 19, 18, 17, 16};

        matrix\_b\_in = '{4, 4, 4, 4, 4};

*repeat*(1) @(negedge clk);

        matrix\_a\_in = '{25, 24, 23, 22, 21};

        matrix\_b\_in = '{5, 5, 5, 5, 5};

*repeat*(1) @(negedge clk);

        valid\_in = 0;

*repeat*(13) @(negedge clk);

*$display*("Test 2 should be done");

*$stop*;

*//===============================================================*

*// Test Case 3 5\*5*

        matrix\_a\_in = '{1, 3, 5, 7, 9};

        matrix\_b\_in = '{5, 4, 3, 2, 1};

        valid\_in = 1;

*repeat*(1) @(negedge clk);

        matrix\_a\_in = '{2, 4, 6, 8, 10};

        matrix\_b\_in = '{10, 9, 8, 7, 6};

*repeat*(1) @(negedge clk);

        matrix\_a\_in = '{11, 13, 15, 17, 19};

        matrix\_b\_in = '{15, 14, 13, 12, 11};

*repeat*(1) @(negedge clk);

        matrix\_a\_in = '{12, 14, 16, 18, 20};

        matrix\_b\_in = '{20, 19, 18, 17, 16};

*repeat*(1) @(negedge clk);

        matrix\_a\_in = '{21, 22, 23, 24, 25};

        matrix\_b\_in = '{25, 24, 23, 22, 21};

*repeat*(1) @(negedge clk);

        valid\_in = 0;

*repeat*(13) @(negedge clk);

*$display*("Test 3 should be done");

*$stop*;

*//===============================================================*

*// Test Case 4 5\*5*

        matrix\_a\_in = '{1, 0, 1, 0, 1};

        matrix\_b\_in = '{1, 2, 3, 4, 5};

        valid\_in = 1;

*repeat*(1) @(negedge clk);

        matrix\_a\_in = '{0, 1, 0, 1, 0};

        matrix\_b\_in = '{5, 4, 3, 2, 1};

*repeat*(1) @(negedge clk);

        matrix\_a\_in = '{1, 1, 1, 1, 1};

        matrix\_b\_in = '{1, 2, 3, 4, 5};

*repeat*(1) @(negedge clk);

        matrix\_a\_in = '{2, 2, 2, 2, 2};

        matrix\_b\_in = '{5, 4, 3, 2, 1};

*repeat*(1) @(negedge clk);

        matrix\_a\_in = '{3, 3, 3, 3, 3};

        matrix\_b\_in = '{1, 2, 3, 4, 5};

*repeat*(1) @(negedge clk);

        valid\_in = 0;

*repeat*(13) @(negedge clk);

*$display*("Test 4 should be done");

*$stop*;

*//===============================================================*

*// Test Case 5 5\*5*

        matrix\_a\_in = '{1, 1, 1, 1, 1};

        matrix\_b\_in = '{1, 2, 3, 4, 5};

        valid\_in = 1;

*repeat*(1) @(negedge clk);

        matrix\_a\_in = '{2, 2, 2, 2, 2};

        matrix\_b\_in = '{6, 7, 8, 9, 10};

*repeat*(1) @(negedge clk);

        matrix\_a\_in = '{3, 3, 3, 3, 3};

        matrix\_b\_in = '{11, 12, 13, 14, 15};

*repeat*(1) @(negedge clk);

        matrix\_a\_in = '{4, 4, 4, 4, 4};

        matrix\_b\_in = '{16, 17, 18, 19, 20};

*repeat*(1) @(negedge clk);

        matrix\_a\_in = '{5, 5, 5, 5, 5};

        matrix\_b\_in = '{21, 22, 23, 24, 25};

*repeat*(1) @(negedge clk);

        valid\_in = 0;

*repeat*(13) @(negedge clk);

*$display*("Test 5 should be done");

*$stop*;

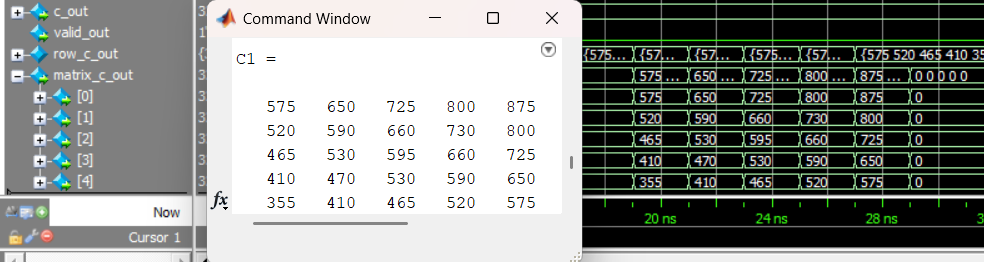
        end

endmodule

# Simulation Output (Questasim):

I used Matlab as a golden model to compare output validation & correctness:

Each snippet Is integrated with Matlab output of same test case:

*Case 1:*

*Case 2:*

A screenshot of a computer

AI-generated content may be incorrect.

*Case 3:*

A screenshot of a computer

AI-generated content may be incorrect.A screenshot of a computer

AI-generated content may be incorrect.

*Case 4:*

*A screenshot of a computer

AI-generated content may be incorrect.Case 5:*

# Vivado Output:

FPGA USED:

## Constraint File for Timing Analysis:

*## Clock signal*

*set\_property* -dict { PACKAGE\_PIN W5   IOSTANDARD LVCMOS33 } [*get\_ports* clk]

*create\_clock* -add -name sys\_clk\_pin -period 10.00 -waveform {0 5} [*get\_ports* clk]

## Elaborated RLT Schematic:

A green circuit board with many wires

AI-generated content may be incorrect.

## Synthesis :

A green machine with many lines

AI-generated content may be incorrect.

Timing Report: A screenshot of a computer

AI-generated content may be incorrect.

Power Report:

A screenshot of a computer

AI-generated content may be incorrect.