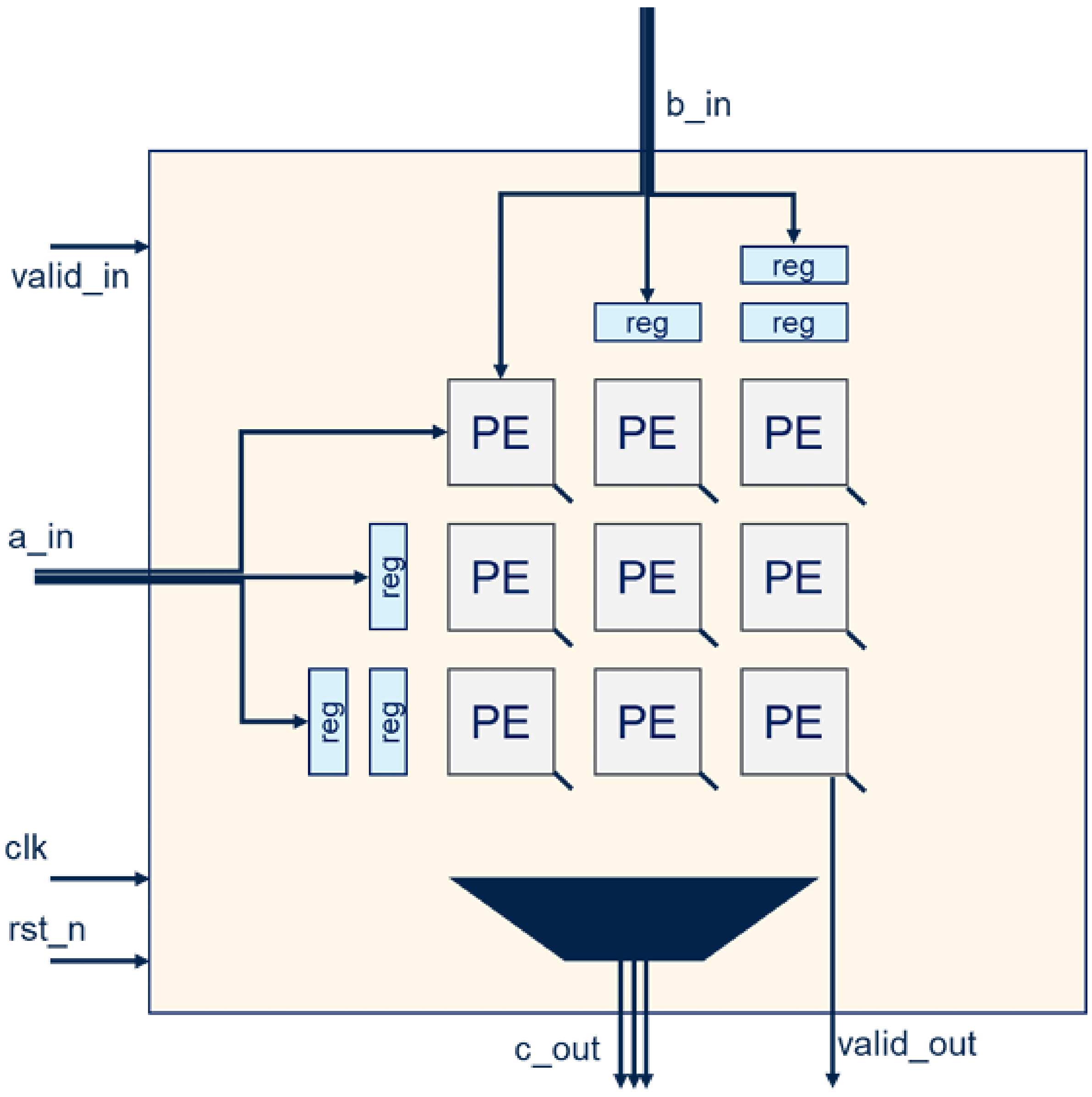
# NxN Systolic Array

**Prepared by:** Mohamed Shaban Moussa

**Email:** mohamedmouse066@gmail.com

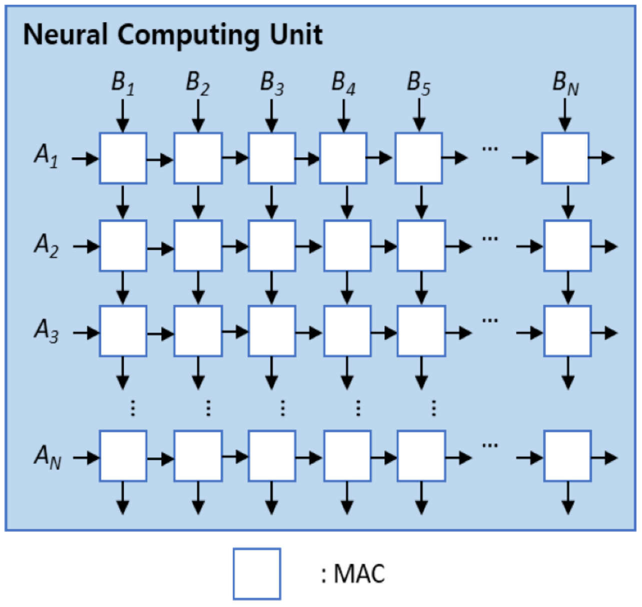
**GitHub Repository:** Press here

- ## Architecture

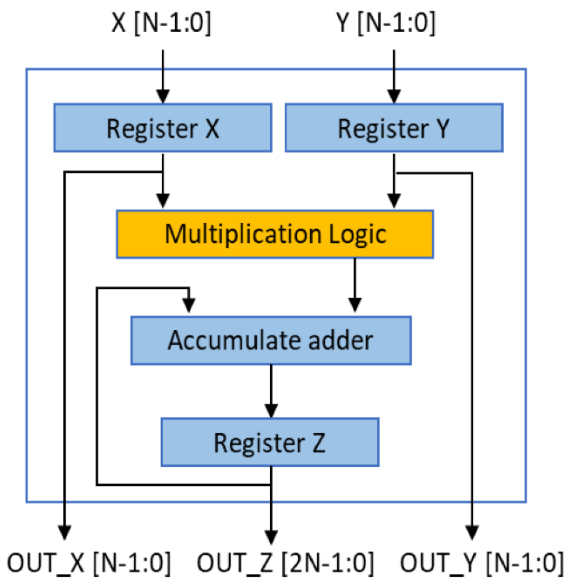


- ## Report Flow
- **Detailed Architecture**
- **Architecture Overview & Module Responsibilities**
- **Challenges Faced During Implementation**
- **Schematic Snapshots**
- **Simulation Results & Test Examples**

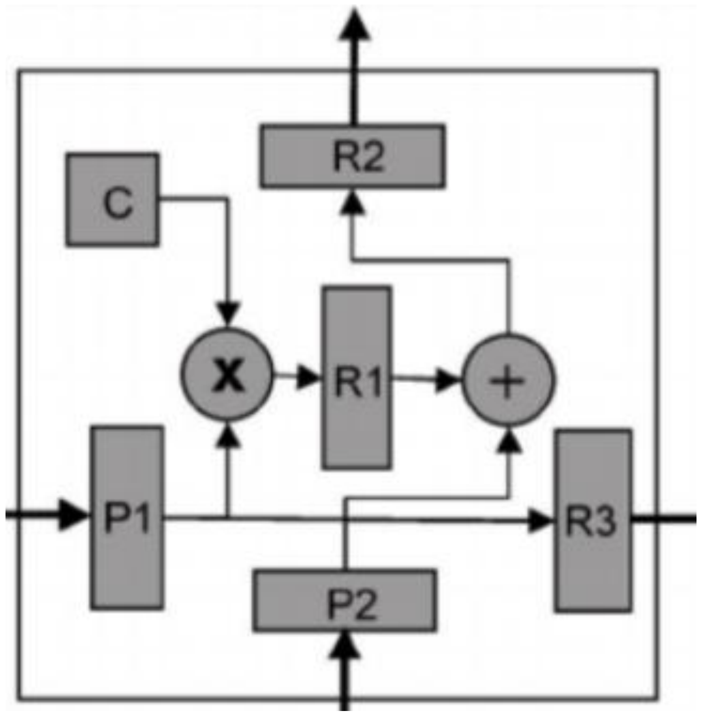
## • Detailed Architecture



(a)



(b)



## 1. Architecture Overview & Module Responsibilities

1. The implemented design is primarily based on **structural modeling**. The entire system is constructed through the hierarchical instantiation of modules, including registers, processing elements (PEs), and control logic.
2. The design features **one counter**: for validating the output.
3. Additionally, the number of **registers** and **PEs (Processing Elements)** is parameterized and depends directly on the selected **matrix size (N_SIZE)**.
4. Each PE operates synchronously, receiving delayed inputs from the left and top, performing multiply-and-accumulate (MAC) operations, and forwarding results to the right and down, forming a systolic computation grid.

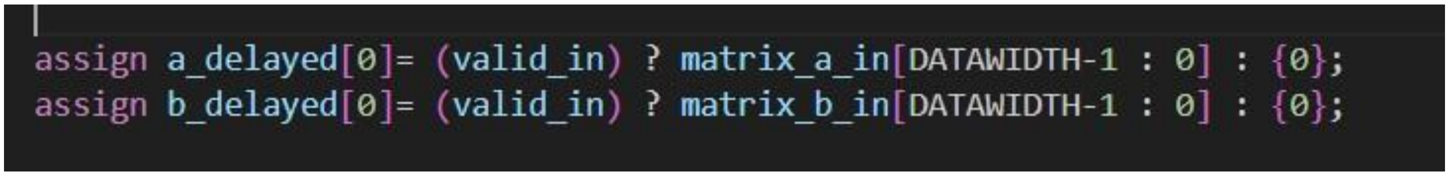
Firstly I have 3 modules In Design

Top Layer Module is systolic_array

This module takes inputs from the user and sends them to the pipelining stage, with a variable number of stages. If Valid-In High
Therefore, it is not suitable to implement a fixed number of registers in the REG module to delay inputs based on the required clock cycles needed for systolic array operation.

**Note:** In the generate loop used to instantiate the REG module, we start from index 1 because the **first element is passed directly to the PE**.
The loop ends when we reach the full size of matrix_a_in and matrix_b_in.

I access the first elements of matrix_a_in and matrix_b_in using a simple method because they **do not enter the pipeline** (don't depend on clk)



```
assign a_delayed[0]= (valid_in) ? matrix_a_in[DATAWIDTH-1 : 0] : {0};
assign b_delayed[0]= (valid_in) ? matrix_b_in[DATAWIDTH-1 : 0] : {0};
```

While designing the pipelining part, I realized that I need a 2D array of registers to implement flexible delay stages for any matrix size.
This allows me to support parameterized systolic arrays efficiently.

So, I used loop-based instantiation

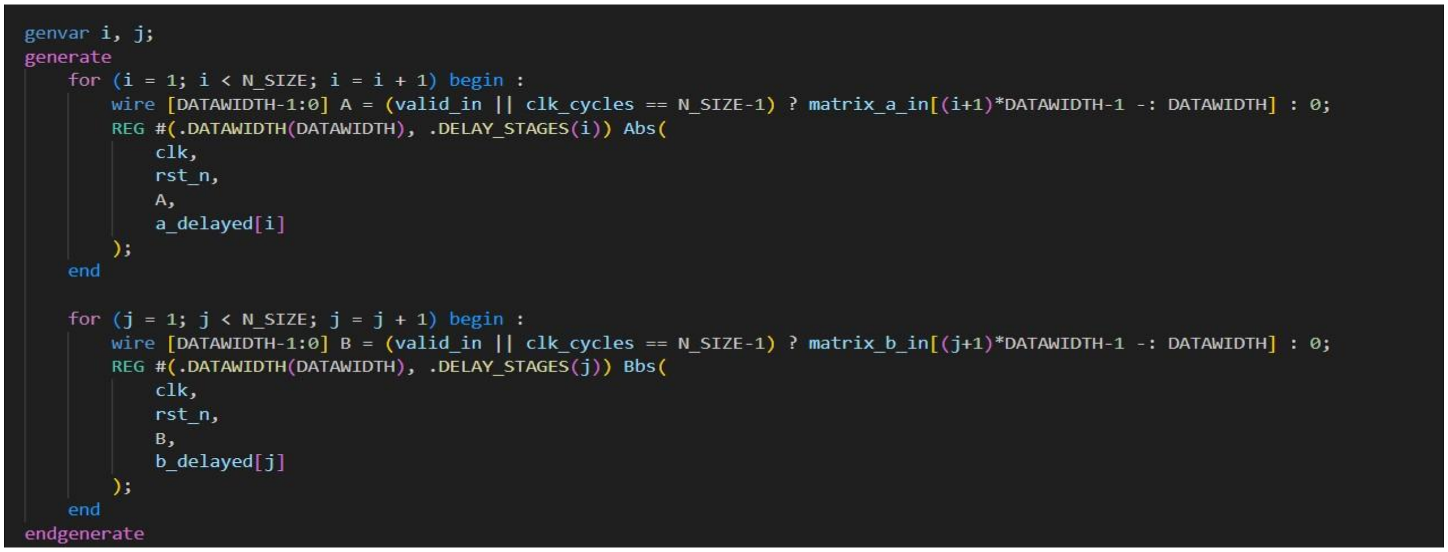
**I overwrite the WIDTH parameter with DATAWIDTH to prevent size mismatches between modules.**

**I also control the delay stages dynamically using the loop index:**

- **The second element in matrix_a_in and matrix_b_in needs 1 register stage.**
- **The third element needs 2 stages, and so on...**
- **The last element requires N_SIZE - 1 delay stages. And I implement part selection by**

matrix_b_in[(i+1)*DATAWIDTH-1 -: DATAWIDTH]

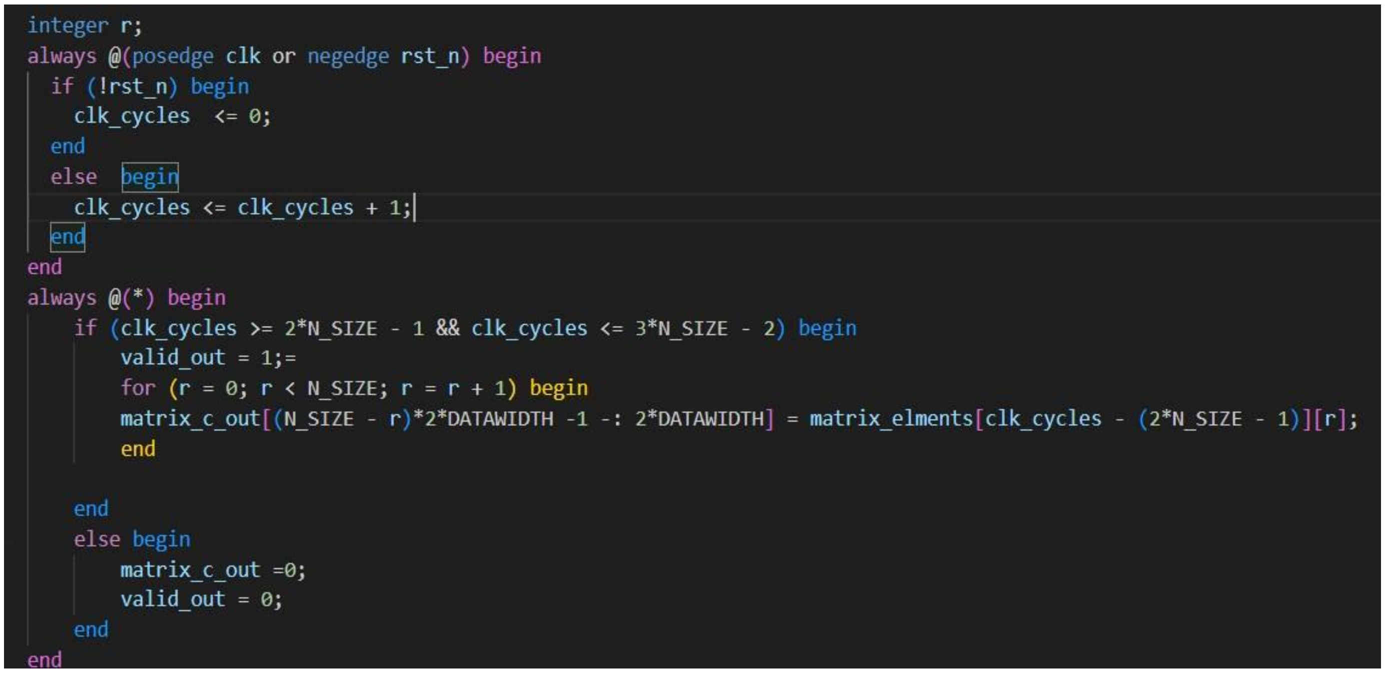
- **means Starting at bit (i+1)*DATAWIDTH – 1 and take DATAWIDTH bits going downward matrix_b_in[(1+1)*16 - 1 -: 16]❼ selects bits [31:16]**
- **After applying suitable pipelining delays to each input,**
  **I instantiate the PE grid to start the accumulation and processing operations through the systolic array.**



```
genvar i, j;
generate
    for (i = 1; i < N_SIZE; i = i + 1) begin :
        wire [DATAWIDTH-1:0] A = (valid_in || clk_cycles == N_SIZE-1) ? matrix_a_in[(i+1)*DATAWIDTH-1 -: DATAWIDTH] : 0;
        REG #(.DATAWIDTH(DATAWIDTH), .DELAY_STAGES(i)) Abs(
            clk,
            rst_n,
            A,
            a_delayed[i]
        );
    end

    for (j = 1; j < N_SIZE; j = j + 1) begin :
        wire [DATAWIDTH-1:0] B = (valid_in || clk_cycles == N_SIZE-1) ? matrix_b_in[(j+1)*DATAWIDTH-1 -: DATAWIDTH] : 0;
        REG #(.DATAWIDTH(DATAWIDTH), .DELAY_STAGES(j)) Bbs(
            clk,
            rst_n,
            B,
            b_delayed[j]
        );
    end
endgenerate
```

- **I send the delayed inputs to the PE griding, and after (3N - 2) clock cycles the correct result of the matrix multiplication appears in the matrix_elments.**
- **I then capture the output rows at specific times.**
  **For example, the first row completes at clock cycle (2N - 1)**

  **The final operation in the systolic_array module is capturing the output matrix_out_c**
- **and controlling the valid_out signal using a clk_cycles counter.**
- **This counter starts counting from the moment the user inputs the first data.**
- **It resets to zero either when rst_n is activated (low) or when the counter reaches the target cycle (3N - 2).in this Cycle we Capture Last Row for matrix_out_c**



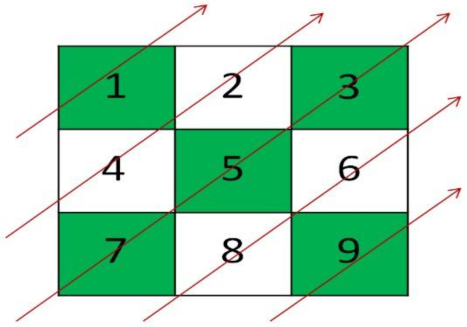
```
integer r;
always @(posedge clk or negedge rst_n) begin
  if (!rst_n) begin
    clk_cycles  <= 0;
  end
  else  begin
    clk_cycles <= clk_cycles + 1;
  end
end
always @(*) begin
    if (clk_cycles >= 2*N_SIZE - 1 && clk_cycles <= 3*N_SIZE - 2) begin
        valid_out = 1;=
        for (r = 0; r < N_SIZE; r = r + 1) begin
        matrix_c_out[(N_SIZE - r)*2*DATAWIDTH -1 -: 2*DATAWIDTH] = matrix_elments[clk_cycles - (2*N_SIZE - 1)][r];
        end

    end
    else begin
        matrix_c_out =0;
        valid_out = 0;
    end
end
```

- **I check whether the clk_counter has reached the minimum target cycle to begin capturing the output.**
  **The first row of matrix_out_c is ready at cycle (2N - 1), the second at 2N, the third at 2N + 1, and so on.**
- **Since the systolic algorithm propagates data diagonally, the output matrix becomes ready row by row, starting from the main diagonal.**
- **The last row is ready at cycle (3N - 2).**
- **During this range of clock cycles, I assert the valid_out signal and begin capturing rows into the output matrix matrix_out_c.**

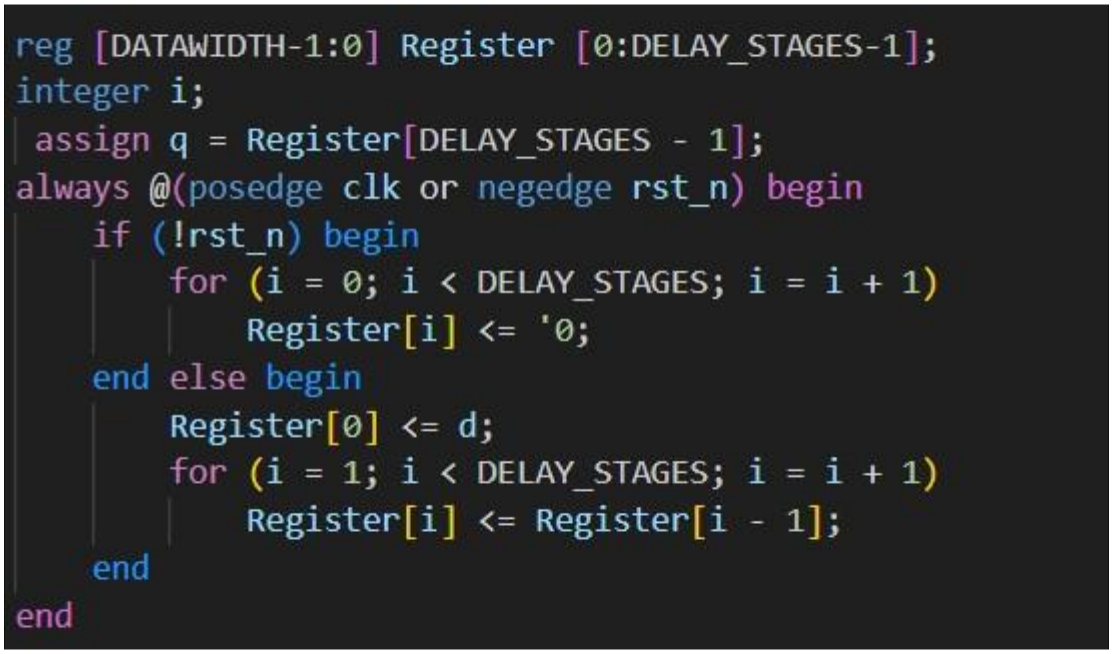
**For more illustrations**

**At first End 1, second End 4,2**

**Third end 7,5,3 , fourth end 6,8**



- ## Secondly the Pipelining Module REG



```
reg [DATAWIDTH-1:0] Register [0:DELAY_STAGES-1];
integer i;
 assign q = Register[DELAY_STAGES - 1];
always @(posedge clk or negedge rst_n) begin
    if (!rst_n) begin
        for (i = 0; i < DELAY_STAGES; i = i + 1)
            Register[i] <= '0;
    end else begin
        Register[0] <= d;
        for (i = 1; i < DELAY_STAGES; i = i + 1)
            Register[i] <= Register[i - 1];
    end
end
```

- **This module receives data and a control signal (valid_in) from the top layer. The input data is only processed when valid_in is high, otherwise, the input is blocked.**

**The number of delay stages is passed as an overwritten parameter from the top layer, making the module highly configurable.**
**This allows flexible pipelining to support parametric systolic array architectures, where each input may require a different number of delay stages.**

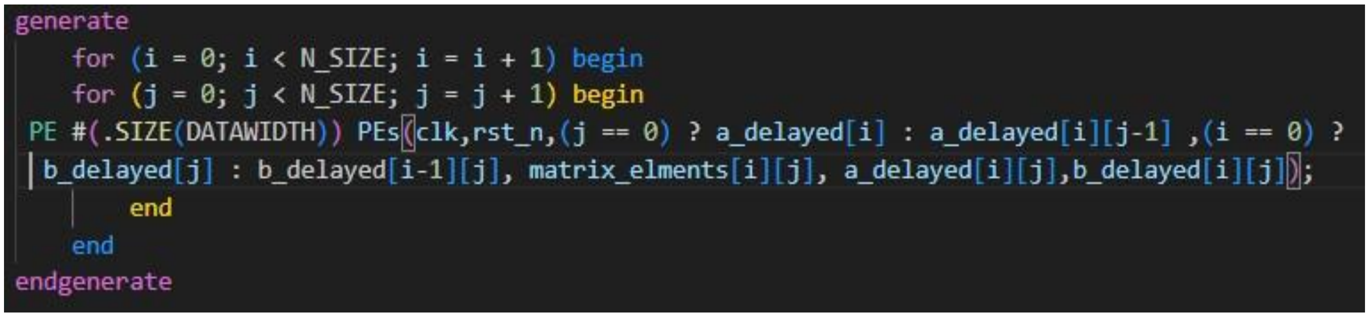
**Register[0] ← in**

- **This line stores the input in into the first register stage.**
- **It initializes the pipeline with the new input value on each clock cycle.**

**The for loop**

- **It shifts the data through the remaining registers.**
- **For each index i from 1 to DELAY_STAGES - 1, the register at position i gets the value from the previous stage Register[i-1].**
- **This effectively passes the input value along the registers introducing a delay.**

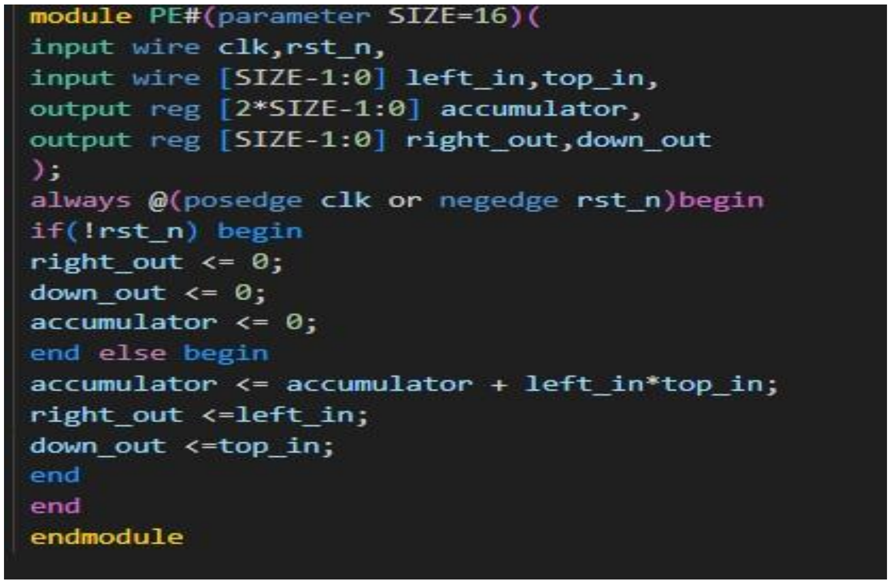
## Third part PEs_GRID



```
generate
    for (i = 0; i < N_SIZE; i = i + 1) begin
    for (j = 0; j < N_SIZE; j = j + 1) begin
 PE #(.SIZE(DATAWIDTH)) PEs(clk,rst_n,(j == 0) ? a_delayed[i] : a_delayed[i][j-1] ,(i == 0) ?
 b_delayed[j] : b_delayed[i-1][j], matrix_elments[i][j], a_delayed[i][j],b_delayed[i][j]);
        end
    end
endgenerate
```

- **The systolic grid takes a parameter N_size and builds a square grid of PEs (Processing Elements).**
- **Grid Instantiation:**
  1. **Inputs a_delayed are connected as columns — one value per row and flow from from left to right.**
  2. **Inputs b_delayed are connected as rows — one value per column and flow from top to bottom**
  3. **Iteration is done from index 0 to N_size- 1 in both directions.**
- **PE Interconnection:**
  1. **Each PE gets input from the top and left.**
  2. **Each PE sends its output to the right neighbor and bottom neighbor, forming a 2D mesh. (Controlled By indexing)**

## Last Module PE



```
module PE#(parameter SIZE=16)(
input wire clk,rst_n,
input wire [SIZE-1:0] left_in,top_in,
output reg [2*SIZE-1:0] accumulator,
output reg [SIZE-1:0] right_out,down_out
);
always @(posedge clk or negedge rst_n)begin
if(!rst_n) begin
right_out <= 0;
down_out <= 0;
accumulator <= 0;
end else begin
accumulator <= accumulator + left_in*top_in;
right_out <=left_in;
down_out <=top_in;
end
end
endmodule
```

- **accumulator: Accumulates the sum of products over multiple cycles.**
- **The PE continues forwarding data even after accumulation finishes. This is crucial for down PEs to receive the correct values.**

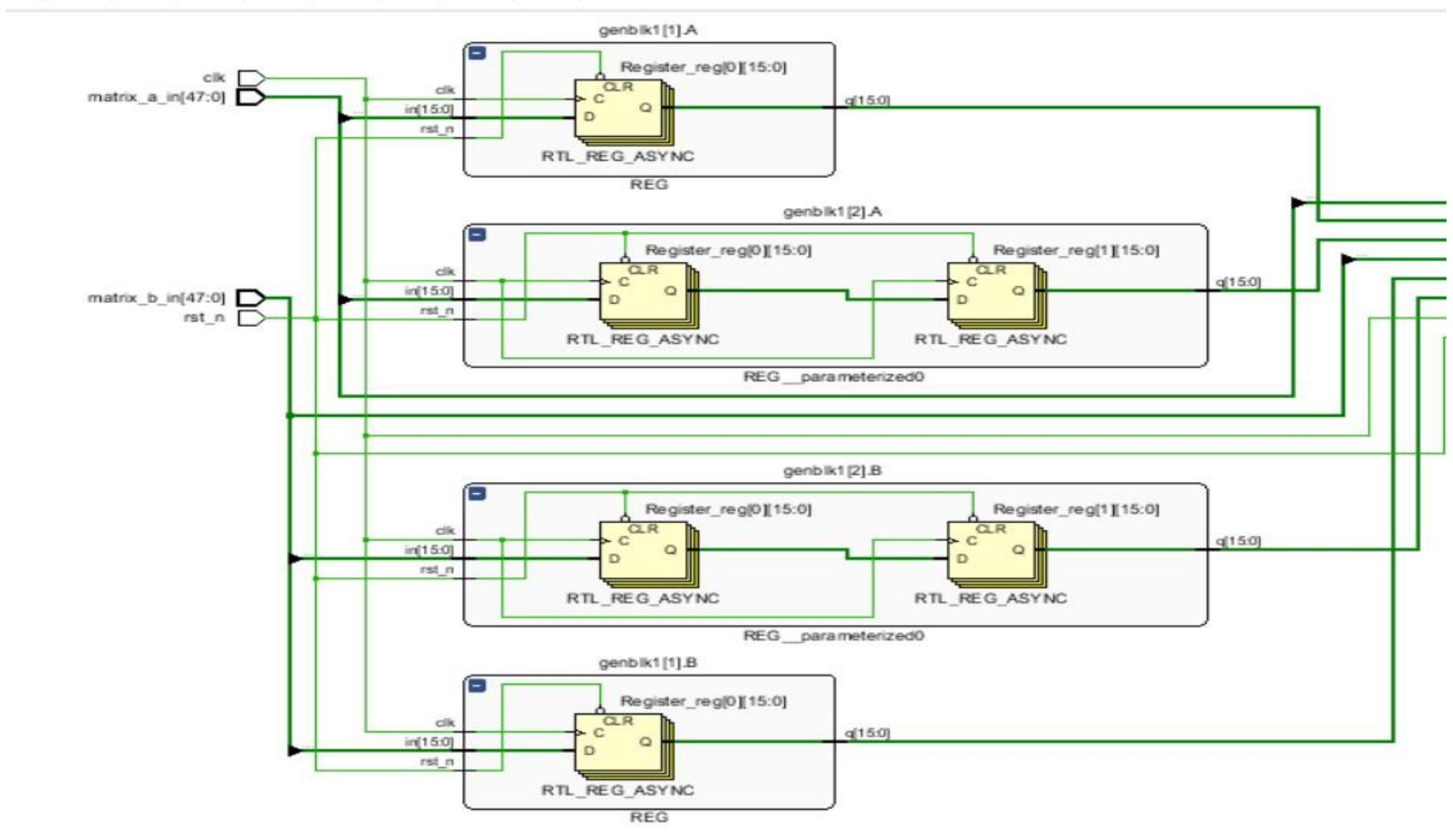
## *2-Challenges Faced During Implementation*

- **In the original design, when N_SIZE = 5, we needed 20 separate register instantiations for each input path.**
- **This manual instantiation led to:**
  1. **Code duplication**
  2. **Poor scalability for different N_SIZE**
  3. **Inefficient resource usage**
  4. **Harder maintenance and debugging**

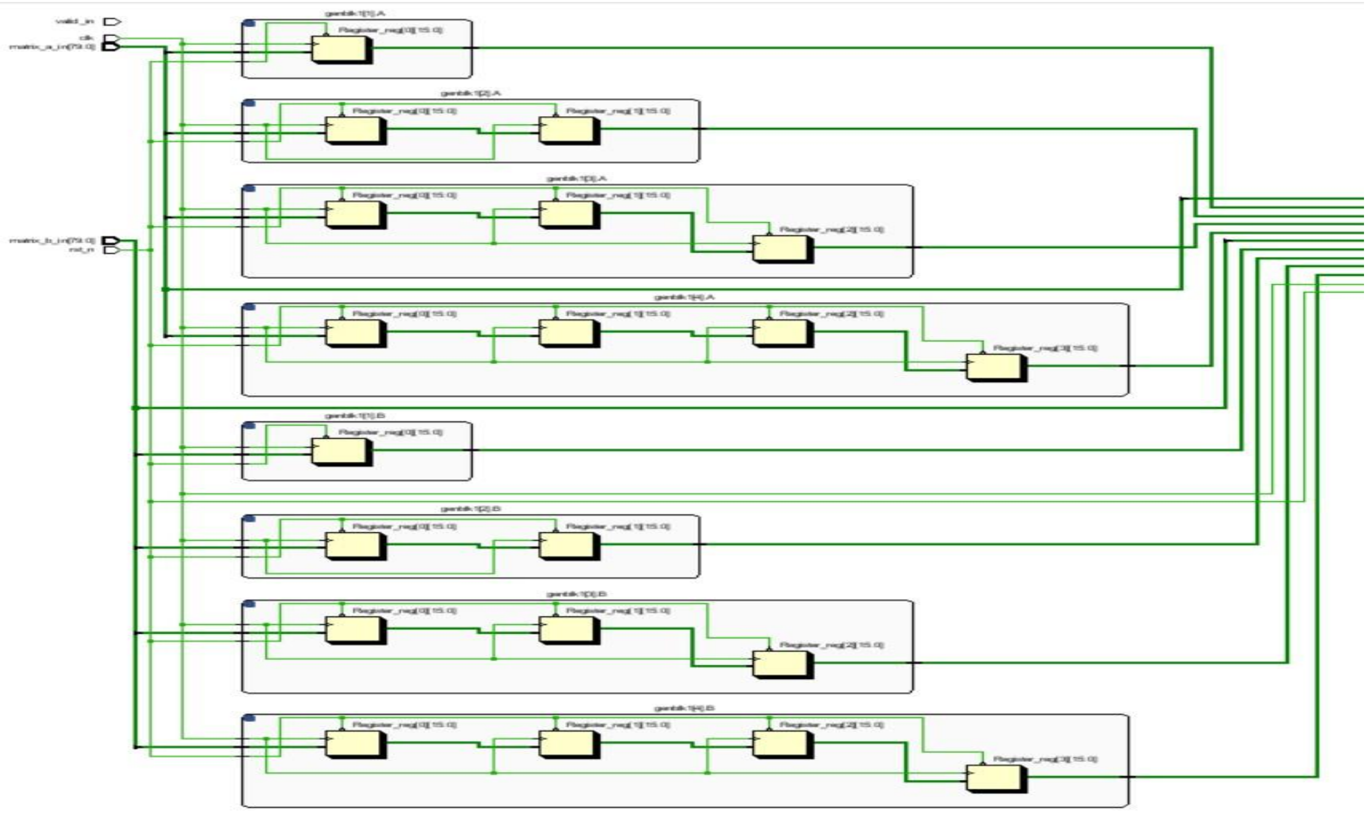
## The Solution Was

1. **Use a single Verilog module with a parameterized number of delay stages.**
2. **Implement the delay line using a register array, where:**
3. **The depth (number of stages) is passed as a parameter**
4. **Data shifts on each clock cycle, forming a pipeline**

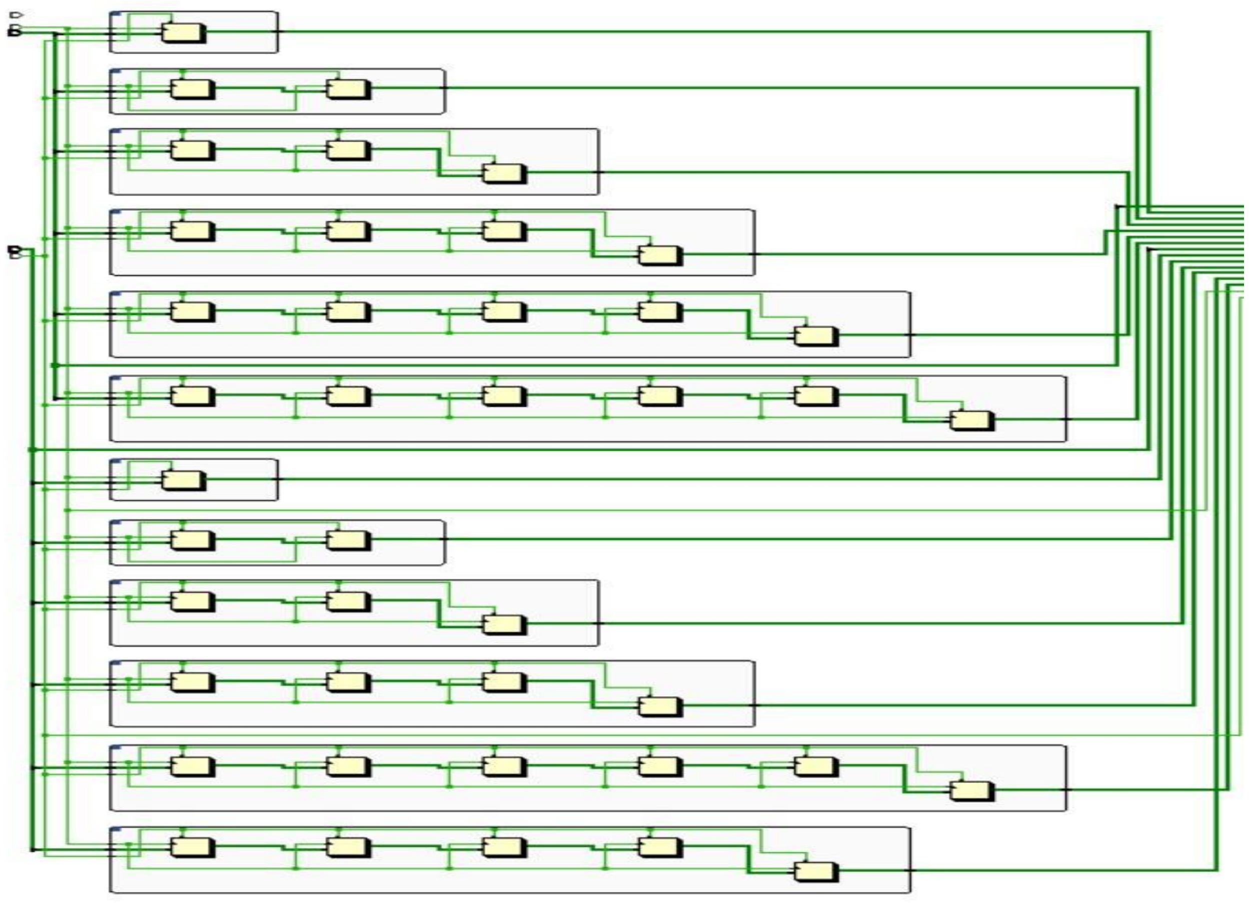
And I get successed **PIPE_LINING WHEN N_SIZE=3**



## PIPE_LINING WHEN N_SIZE=5



## PIPE_LINING WHEN N_SIZE=7



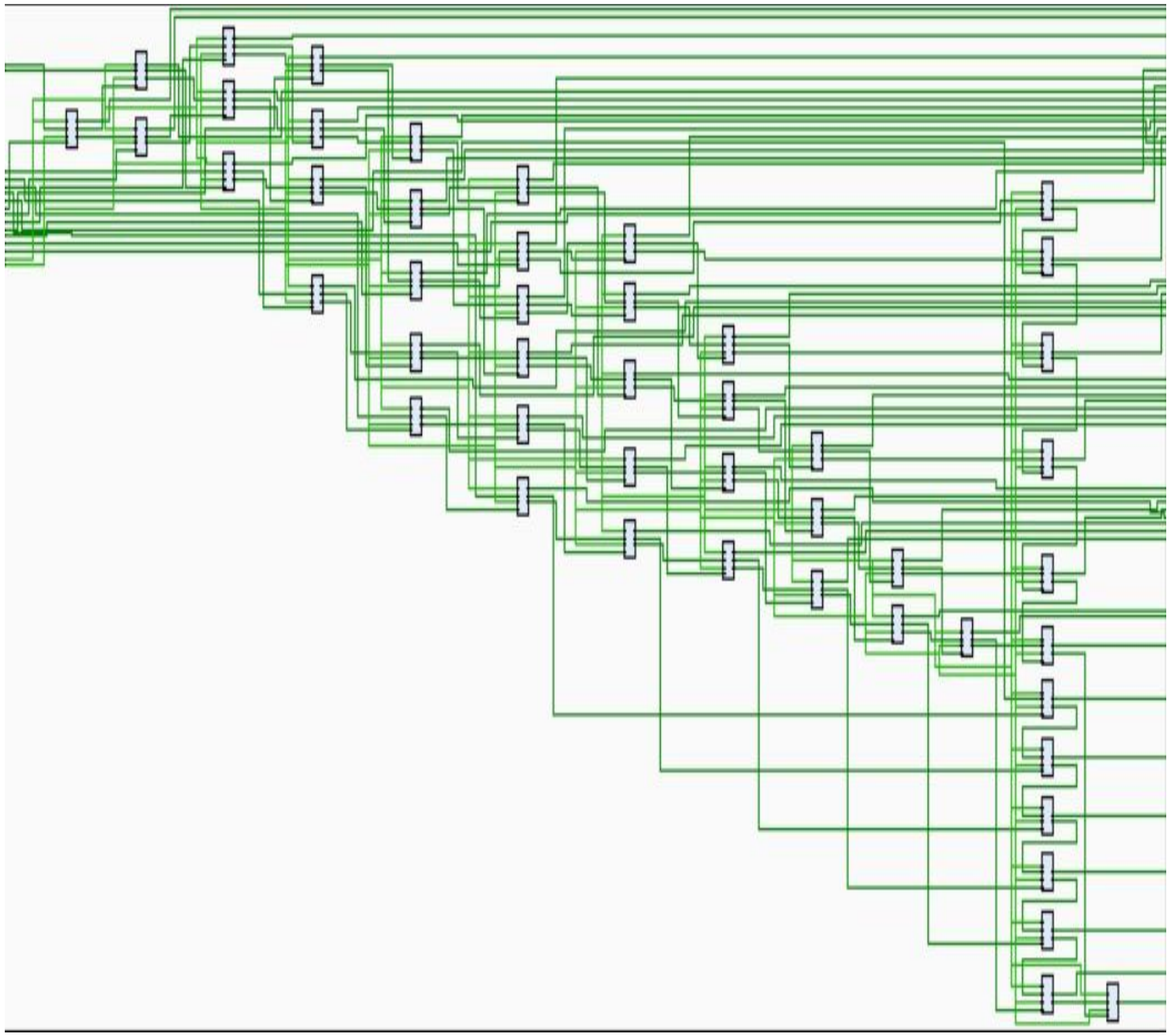
**2_To implement a scalable systolic array with a parameterizable number of Processing Elements (PEs), directly instantiating each PE manually becomes not efficient**

**To solve this, I designed a dedicated Grid Generate that dynamically builds a square grid of size N_SIZE × N_SIZE, where N_SIZE is a top-level parameter representing the matrix dimension.**

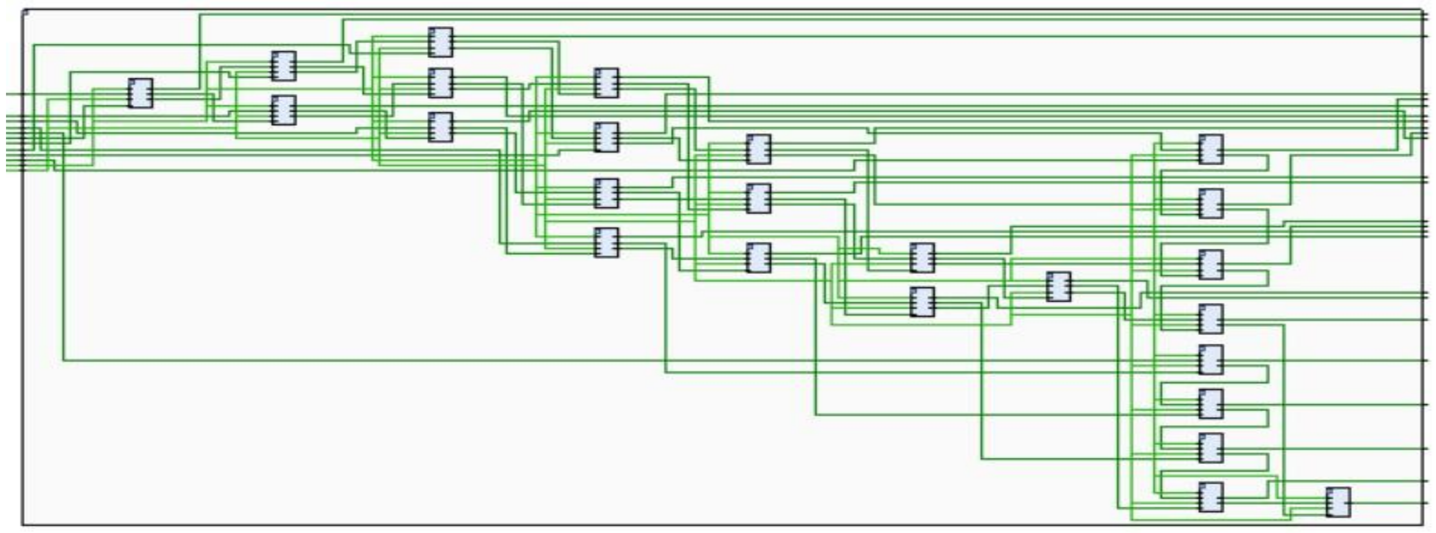
**This module:**

- **Uses nested generate loops to instantiate PEs automatically based on N_SIZE.**
- **Handles 2D data flow across the grid: each PE receives input from the top and left and passes results to the bottom and right neighbors.**

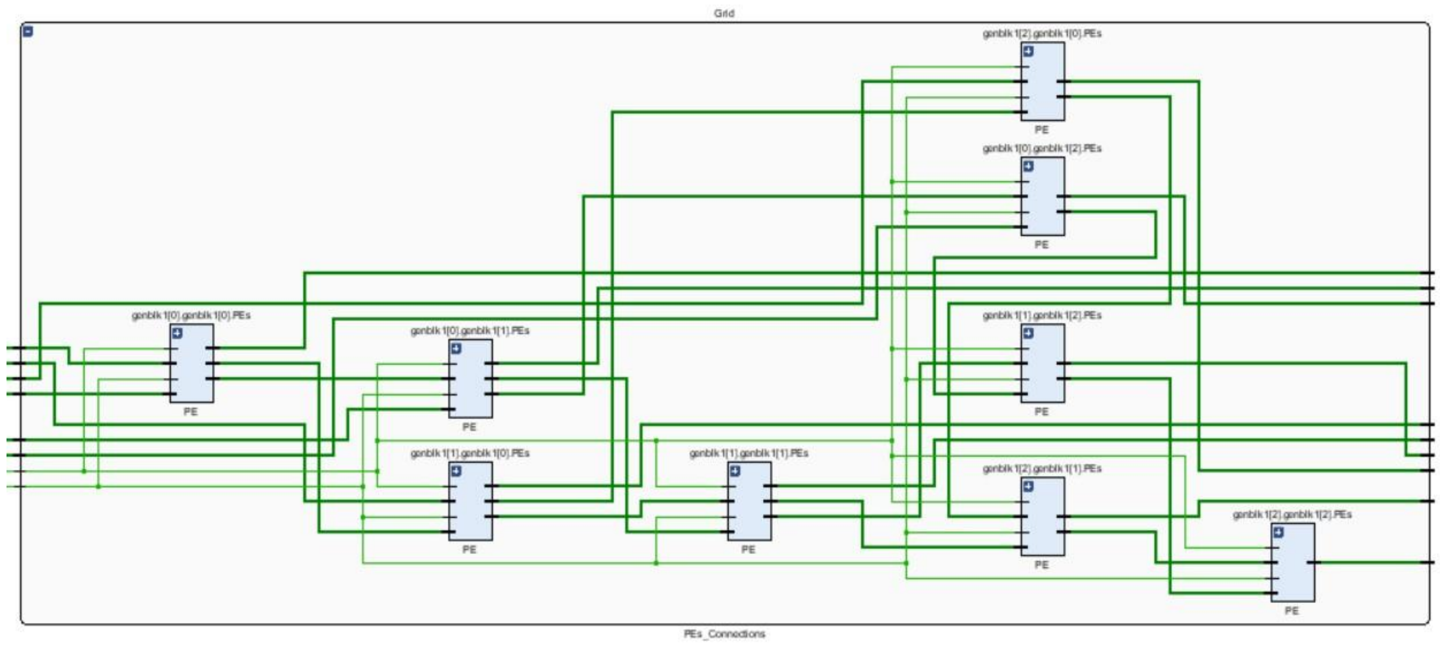
## PEs_Grid_When N_SIZE=7 is 49 PE



## PEs_Grid_When N_SIZE=5 is 25 PE



## PEs_Grid_When N_SIZE=3 is 9 PE

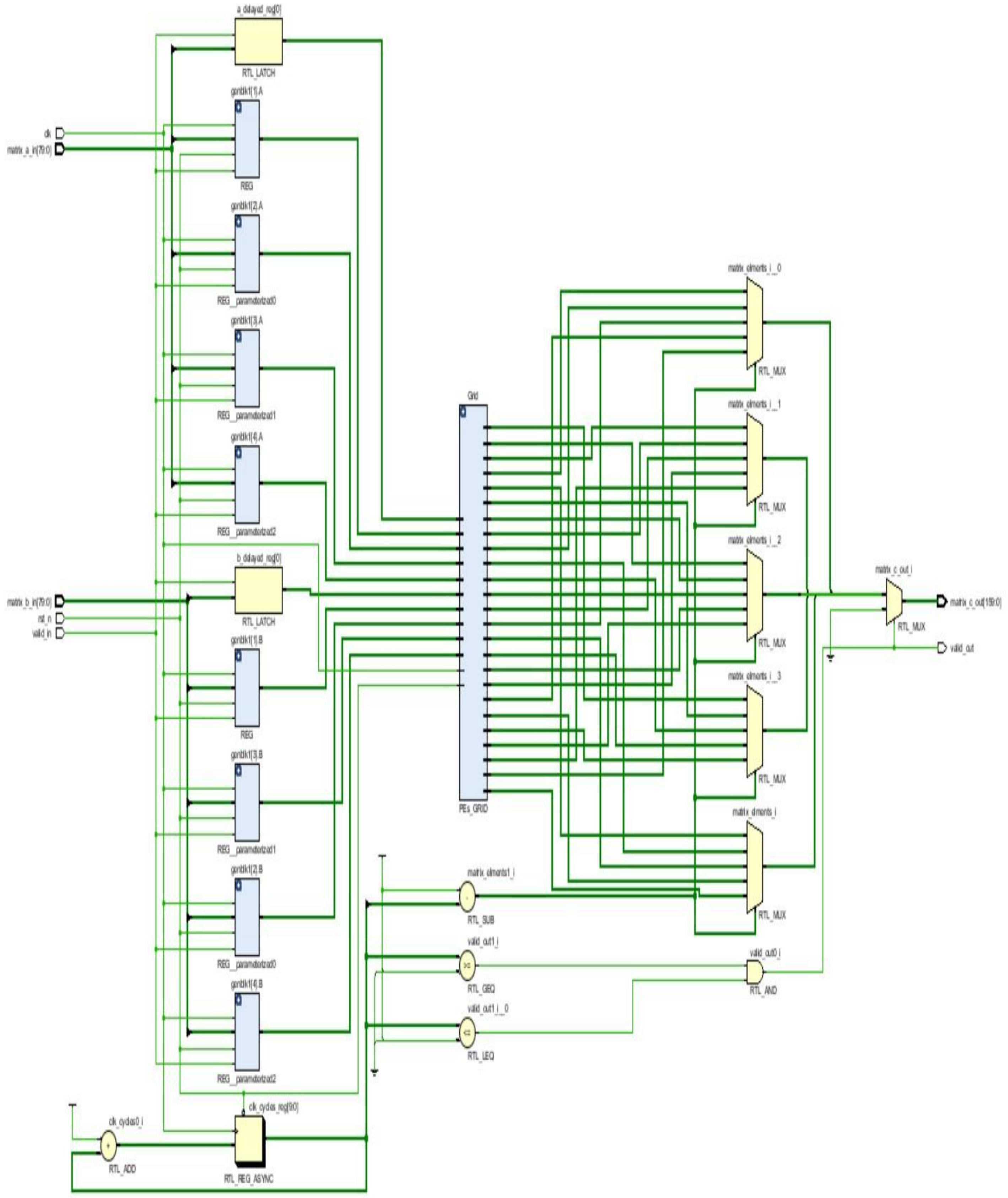

Grid
genblk1[2].genblk1[0].PEs
PE
genblk1[0].genblk1[2].PEs
PE
genblk1[0].genblk1[0].PEs
PE
genblk1[0].genblk1[1].PEs
PE
genblk1[1].genblk1[2].PEs
PE
genblk1[1].genblk1[0].PEs
PE
genblk1[1].genblk1[1].PEs
PE
genblk1[2].genblk1[1].PEs
PE
genblk1[2].genblk1[2].PEs
PE
PEs_Connections

**After resolving several integration issues between modules, I was able to fully parameterize the entire systolic array design. This included:**

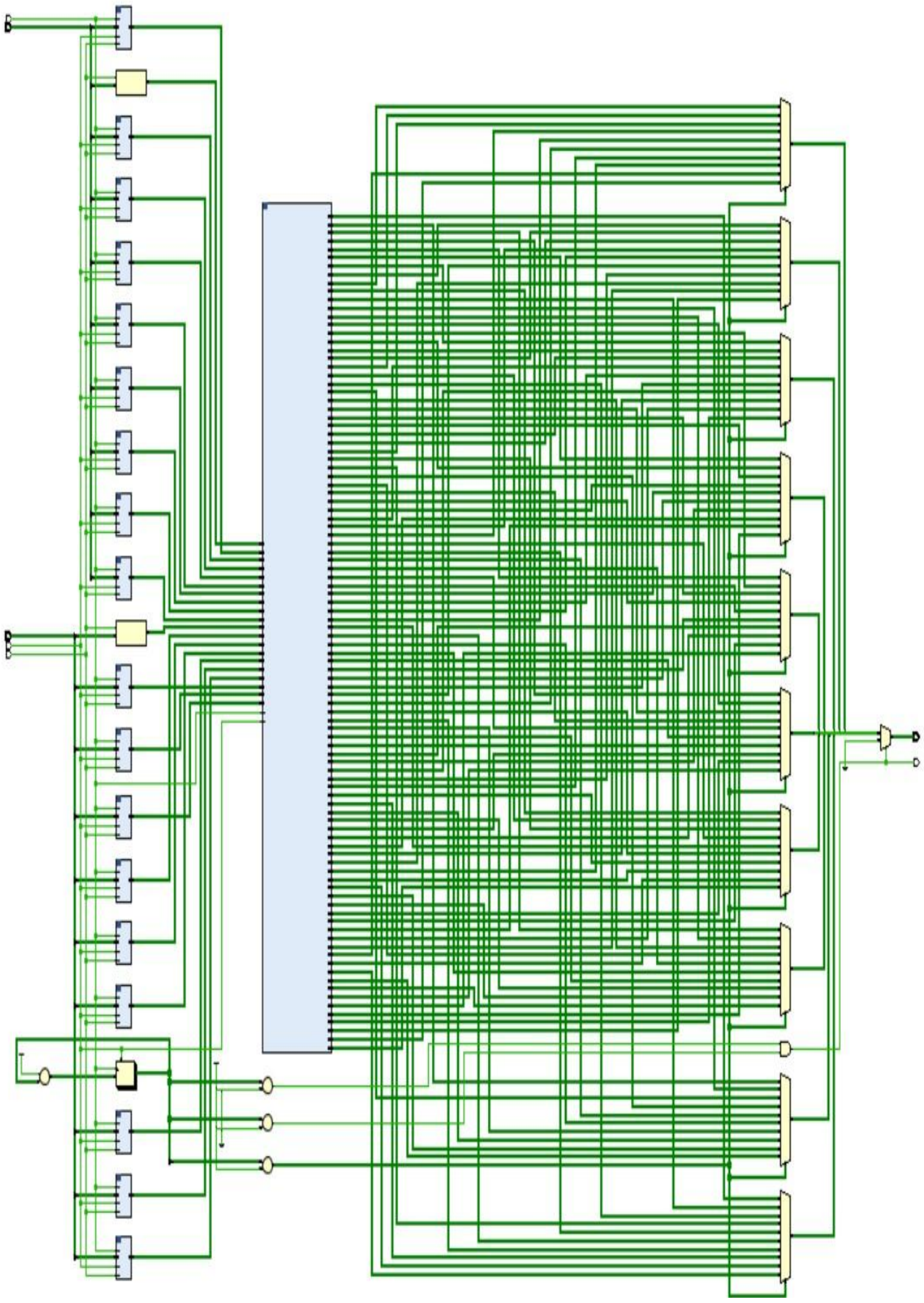
1. **Making the number of delay registers configurable based on matrix size.**
2. **Automatically generating the required number of PEs using nested generate loops.**
3. **Ensuring signal widths, pipeline stages, and accumulation control signals scale correctly with N_SIZE**

- *Schematic Snapshots*

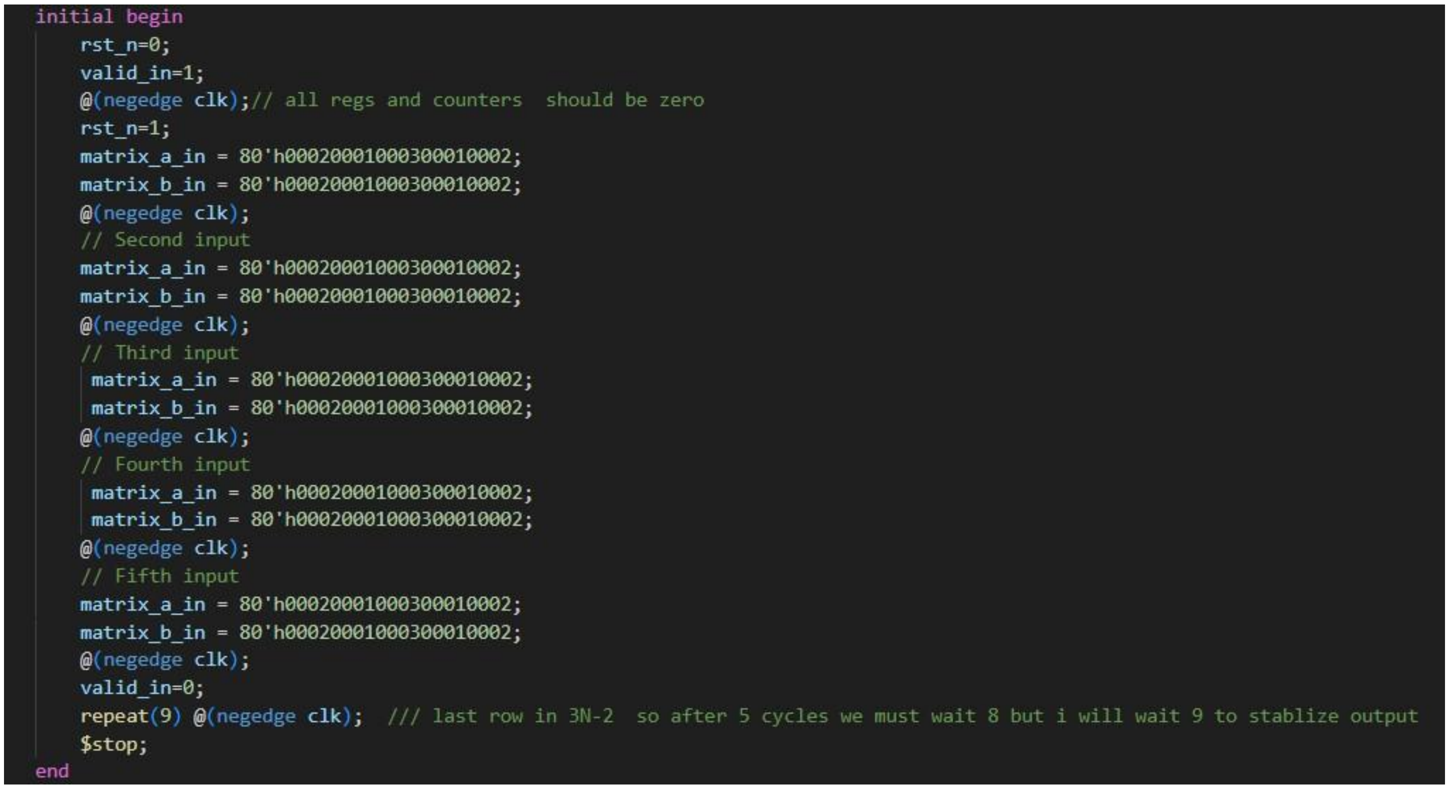
**Full Schematic When N =5**



## Full Schematic When N =10

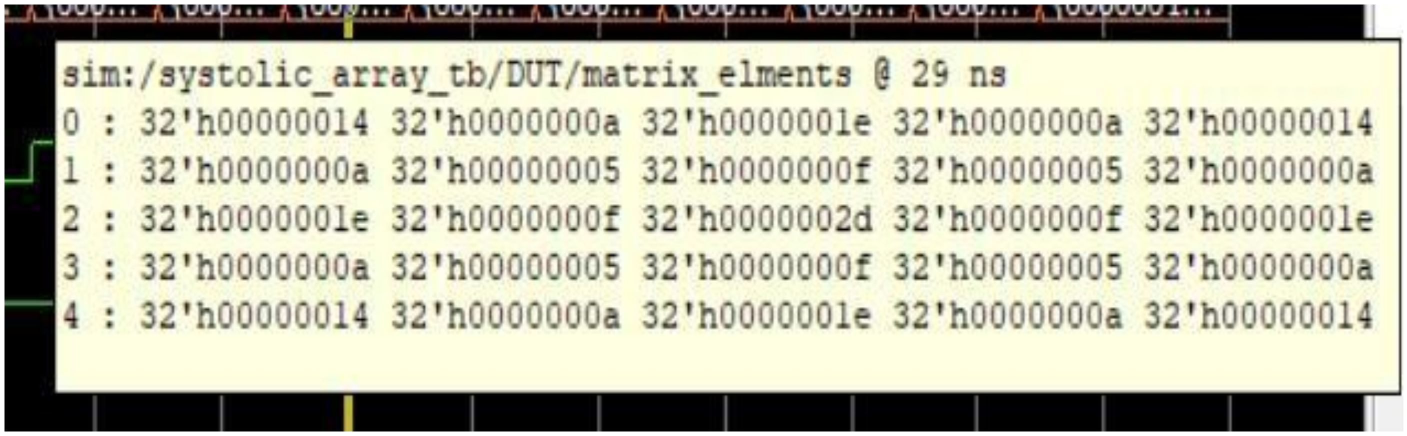


## *4-Simulation Results & Test Examples Test on 5*5 Matrix First test*



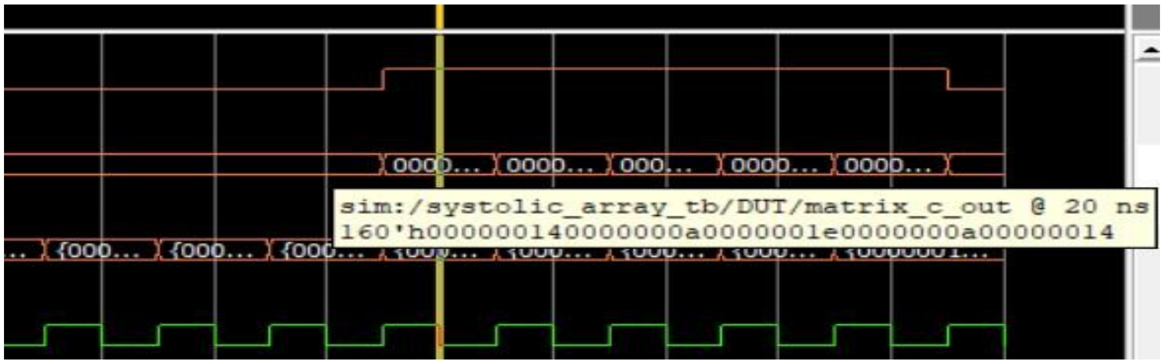
```
initial begin
    rst_n=0;
    valid_in=1;
    @(negedge clk);// all regs and counters  should be zero
    rst_n=1;
    matrix_a_in = 80'h00020001000300010002;
    matrix_b_in = 80'h00020001000300010002;
    @(negedge clk);
    // Second input
    matrix_a_in = 80'h00020001000300010002;
    matrix_b_in = 80'h00020001000300010002;
    @(negedge clk);
    // Third input
     matrix_a_in = 80'h00020001000300010002;
     matrix_b_in = 80'h00020001000300010002;
    @(negedge clk);
    // Fourth input
     matrix_a_in = 80'h00020001000300010002;
     matrix_b_in = 80'h00020001000300010002;
    @(negedge clk);
    // Fifth input
    matrix_a_in = 80'h00020001000300010002;
    matrix_b_in = 80'h00020001000300010002;
    @(negedge clk);
    valid_in=0;
    repeat(9) @(negedge clk);  /// last row in 3N-2  so after 5 cycles we must wait 8 but i will wait 9 to stablize output
    $stop;
end
```

## *Complete true Output Matrix in Hex*



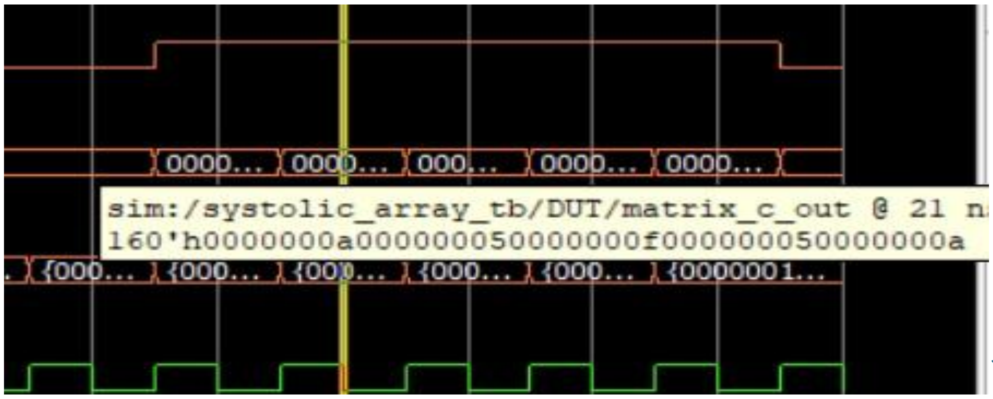
```
sim:/systolic_array_tb/DUT/matrix_elments @ 29 ns
0 : 32'h00000014 32'h0000000a 32'h0000001e 32'h0000000a 32'h00000014
1 : 32'h0000000a 32'h00000005 32'h0000000f 32'h00000005 32'h0000000a
2 : 32'h0000001e 32'h0000000f 32'h0000002d 32'h0000000f 32'h0000001e
3 : 32'h0000000a 32'h00000005 32'h0000000f 32'h00000005 32'h0000000a
4 : 32'h00000014 32'h0000000a 32'h0000001e 32'h0000000a 32'h00000014
```

## First Row when Valid_out get high



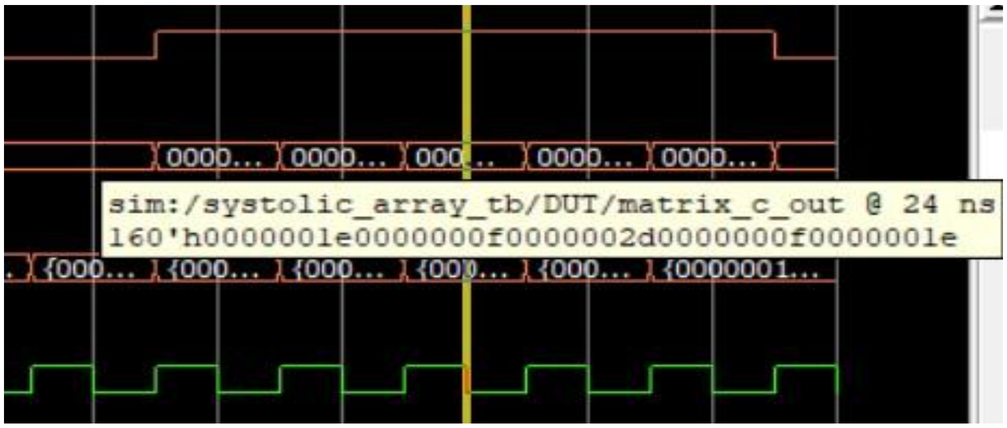
```
sim:/systolic_array_tb/DUT/matrix_c_out @ 20 ns
160'h000000140000000a0000001e0000000a00000014
```

## Second Row when Valid_out high



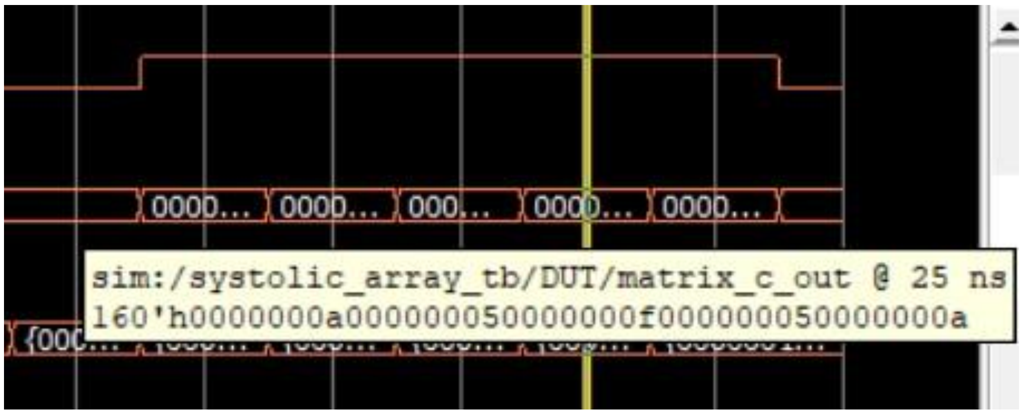
```
sim:/systolic_array_tb/DUT/matrix_c_out @ 21 ns
160'h0000000a000000050000000f000000050000000a
```

## Third Row when Valid_out high



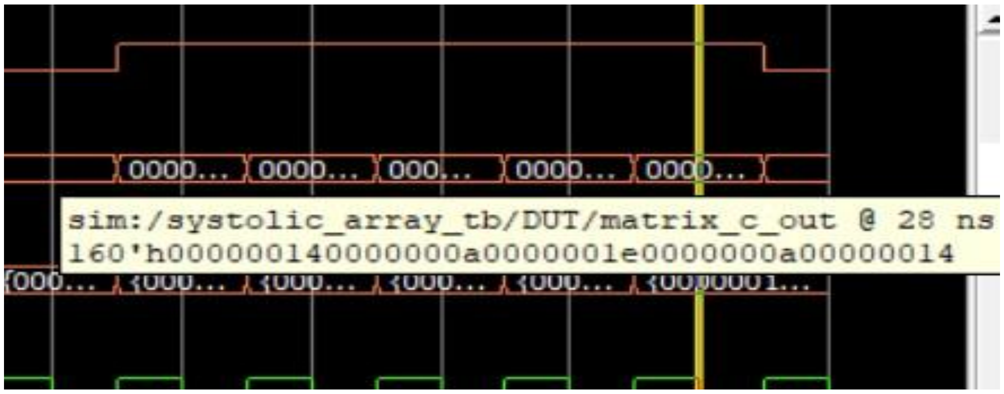
```
sim:/systolic_array_tb/DUT/matrix_c_out @ 24 ns
160'h0000001e0000000f0000002d0000000f0000001e
```

## Fourth Row when Valid_out high



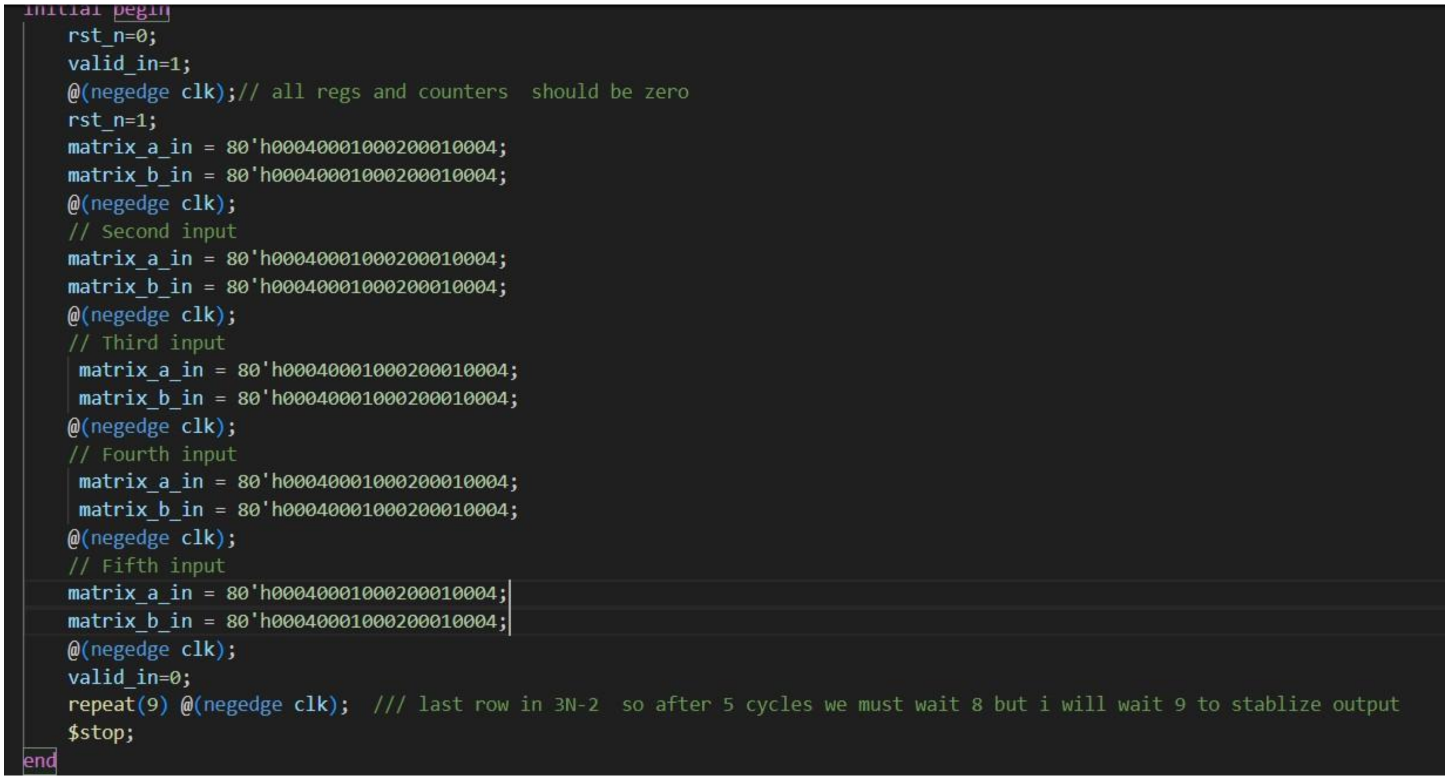
```
sim:/systolic_array_tb/DUT/matrix_c_out @ 25 ns
160'h0000000a000000050000000f000000050000000a
```

## Fifth Row when Valid_out high



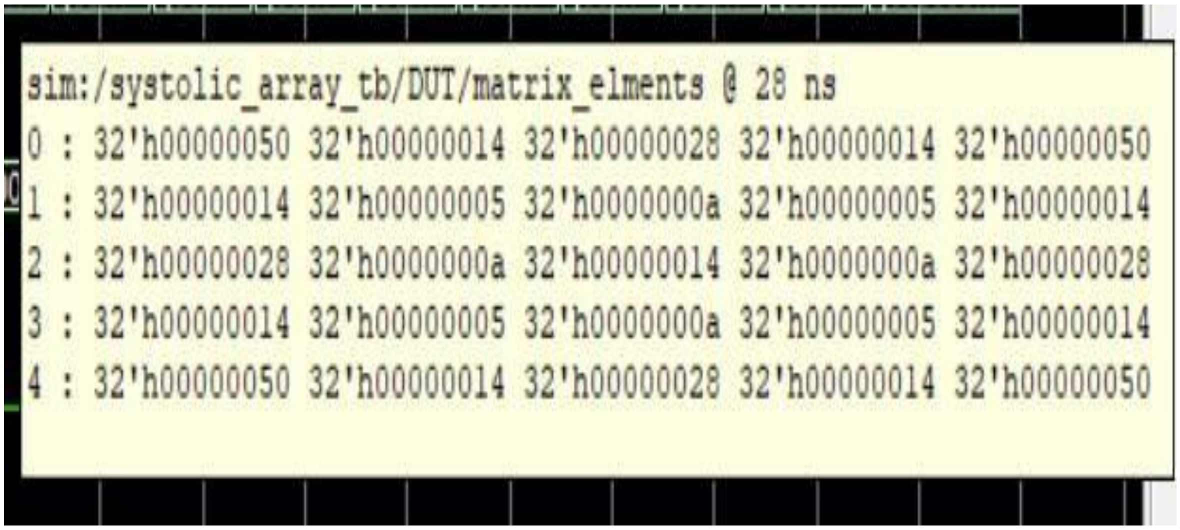
```
sim:/systolic_array_tb/DUT/matrix_c_out @ 28 ns
160'h000000140000000a0000001e0000000a00000014
```

## Second Test



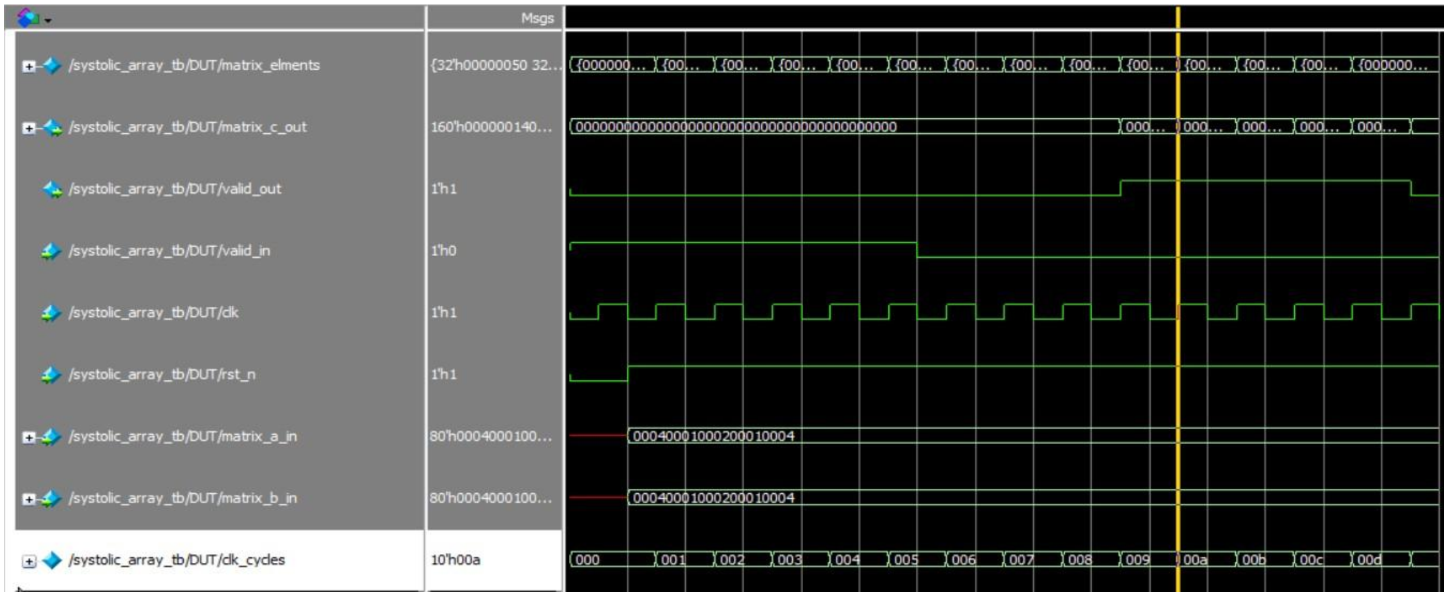
```
initial begin
    rst_n=0;
    valid_in=1;
    @(negedge clk);// all regs and counters  should be zero
    rst_n=1;
    matrix_a_in = 80'h00040001000200010004;
    matrix_b_in = 80'h00040001000200010004;
    @(negedge clk);
    // Second input
    matrix_a_in = 80'h00040001000200010004;
    matrix_b_in = 80'h00040001000200010004;
    @(negedge clk);
    // Third input
     matrix_a_in = 80'h00040001000200010004;
     matrix_b_in = 80'h00040001000200010004;
    @(negedge clk);
    // Fourth input
     matrix_a_in = 80'h00040001000200010004;
     matrix_b_in = 80'h00040001000200010004;
    @(negedge clk);
    // Fifth input
    matrix_a_in = 80'h00040001000200010004;
    matrix_b_in = 80'h00040001000200010004;
    @(negedge clk);
    valid_in=0;
    repeat(9) @(negedge clk);  /// last row in 3N-2  so after 5 cycles we must wait 8 but i will wait 9 to stablize output
    $stop;
end
```

## Full True_Matrix

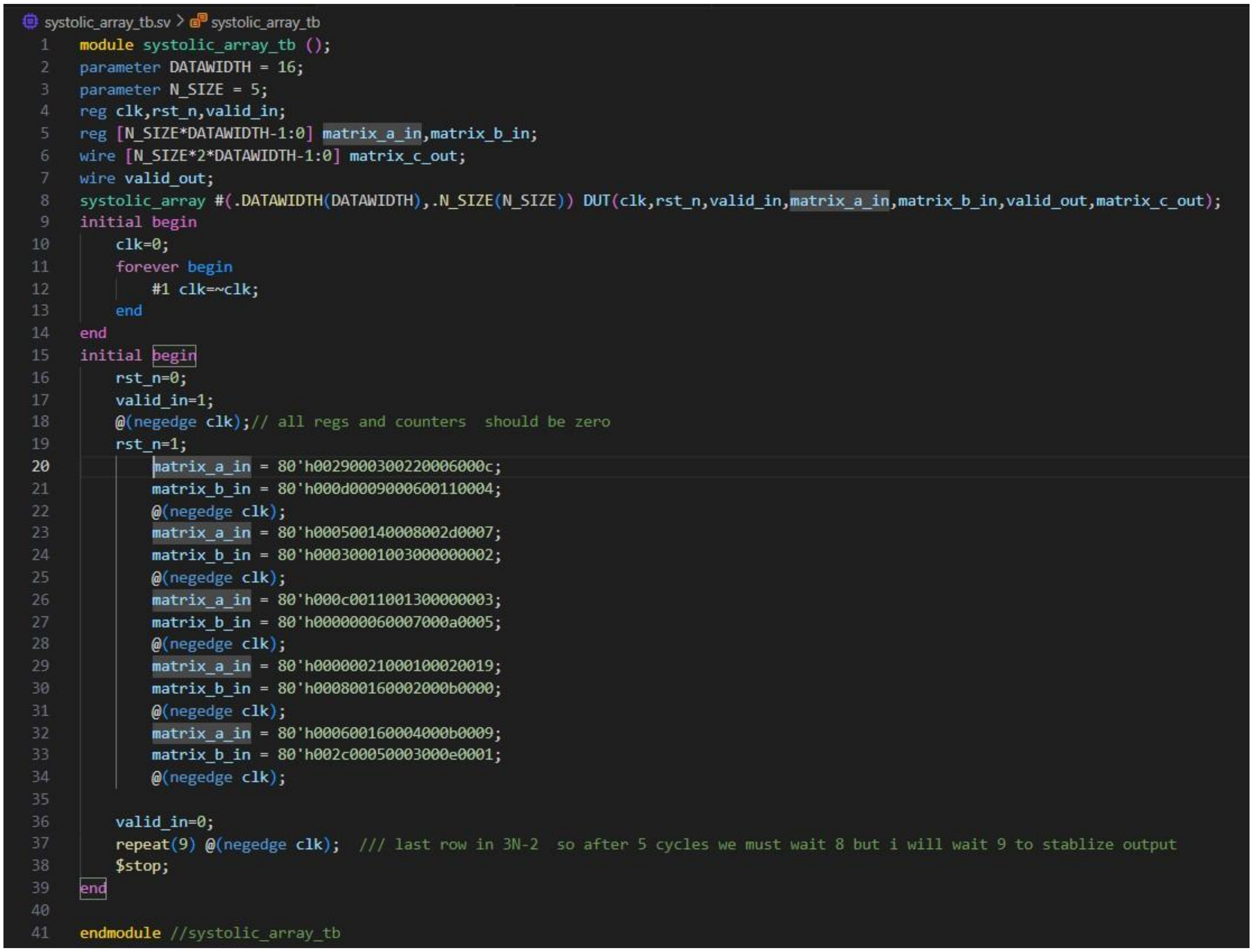


```
sim:/systolic_array_tb/DUT/matrix_elments @ 28 ns
0 : 32'h00000050 32'h00000014 32'h00000028 32'h00000014 32'h00000050
1 : 32'h00000014 32'h00000005 32'h0000000a 32'h00000005 32'h00000014
2 : 32'h00000028 32'h0000000a 32'h00000014 32'h0000000a 32'h00000028
3 : 32'h00000014 32'h00000005 32'h0000000a 32'h00000005 32'h00000014
4 : 32'h00000050 32'h00000014 32'h00000028 32'h00000014 32'h00000050
```

- ## Wave


Msgs
/systolic_array_tb/DUT/matrix_elments {32'h00000050 32...
/systolic_array_tb/DUT/matrix_c_out 160'h000000140...
/systolic_array_tb/DUT/valid_out 1'h1
/systolic_array_tb/DUT/valid_in 1'h0
/systolic_array_tb/DUT/clk 1'h1
/systolic_array_tb/DUT/rst_n 1'h1
/systolic_array_tb/DUT/matrix_a_in 80'h0004000100...
/systolic_array_tb/DUT/matrix_b_in 80'h0004000100...
/systolic_array_tb/DUT/dk_cycles 10'h00a
{000000... {00... {00... {00... {00... {00... {00... {00... {00... {00... {00... {00... {00... {000000...
0000000000000000000000000000000000000000 000... 000... 000... 000... 000...
00040001000200010004
00040001000200010004
000 001 002 003 004 005 006 007 008 009 00a 00b 00c 00d

- ## Third Test



systolic_array_tb.sv > systolic_array_tb

```
module systolic_array_tb ();
parameter DATAWIDTH = 16;
parameter N_SIZE = 5;
reg clk,rst_n,valid_in;
reg [N_SIZE*DATAWIDTH-1:0] matrix_a_in,matrix_b_in;
wire [N_SIZE*2*DATAWIDTH-1:0] matrix_c_out;
wire valid_out;
systolic_array #(.DATAWIDTH(DATAWIDTH),.N_SIZE(N_SIZE)) DUT(clk,rst_n,valid_in,matrix_a_in,matrix_b_in,valid_out,matrix_c_out);
initial begin
    clk=0;
    forever begin
        #1 clk=~clk;
    end
end
initial begin
    rst_n=0;
    valid_in=1;
    @(negedge clk);// all regs and counters  should be zero
    rst_n=1;
        matrix_a_in = 80'h0029000300220006000c;
        matrix_b_in = 80'h000d0009000600110004;
        @(negedge clk);
        matrix_a_in = 80'h000500140008002d0007;
        matrix_b_in = 80'h00030001003000000002;
        @(negedge clk);
        matrix_a_in = 80'h000c0011001300000003;
        matrix_b_in = 80'h000000060007000a0005;
        @(negedge clk);
        matrix_a_in = 80'h00000021000100020019;
        matrix_b_in = 80'h000800160002000b0000;
        @(negedge clk);
        matrix_a_in = 80'h000600160004000b0009;
        matrix_b_in = 80'h002c00050003000e0001;
        @(negedge clk);

    valid_in=0;
    repeat(9) @(negedge clk);  /// last row in 3N-2  so after 5 cycles we must wait 8 but i will wait 9 to stablize output
    $stop;
end

endmodule //systolic_array_tb
```

**Matrix_a_in**

**takes From F**

**Matrix_b_in**

**takes From G**

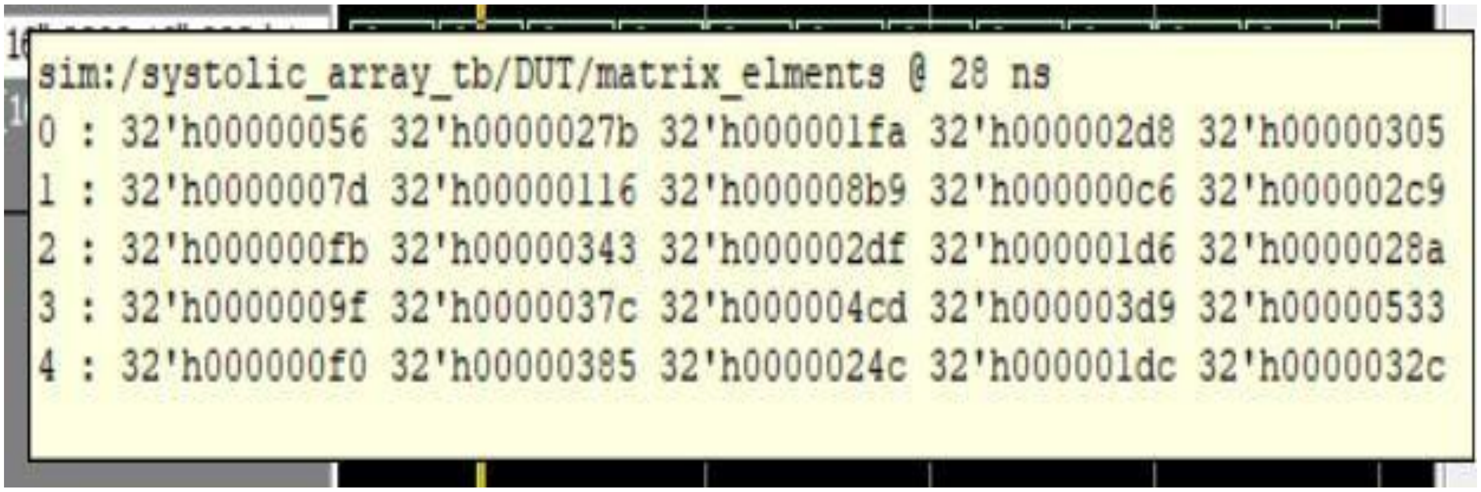
$$F = \begin{bmatrix} 12 & 7 & 3 & 25 & 9 \\ 6 & 45 & 0 & 2 & 11 \\ 34 & 8 & 19 & 1 & 4 \\ 3 & 20 & 17 & 33 & 22 \\ 41 & 5 & 12 & 0 & 6 \end{bmatrix}$$

$$G = \begin{bmatrix} 4 & 17 & 6 & 9 & 13 \\ 2 & 0 & 48 & 1 & 3 \\ 5 & 10 & 7 & 6 & 0 \\ 0 & 11 & 2 & 22 & 8 \\ 1 & 14 & 3 & 5 & 44 \end{bmatrix}$$

**H=F*G**

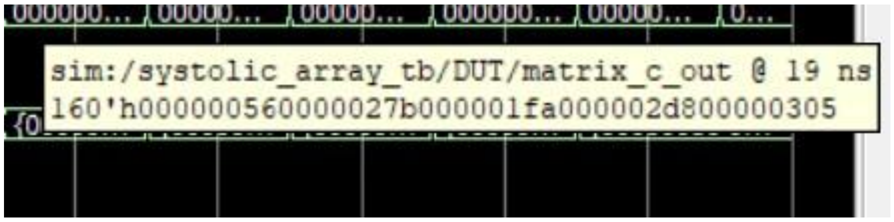
$$H_{\text{hex}} = \begin{bmatrix} 0x56 & 0x27B & 0x1FA & 0x2D8 & 0x305 \\ 0x7D & 0x116 & 0x8B9 & 0xC6 & 0x2C9 \\ 0xFB & 0x343 & 0x2DF & 0x1D6 & 0x28A \\ 0x9F & 0x37C & 0x4CD & 0x3D9 & 0x533 \\ 0xF0 & 0x385 & 0x24C & 0x1DC & 0x32C \end{bmatrix}$$

- **Answer After 13 edge**



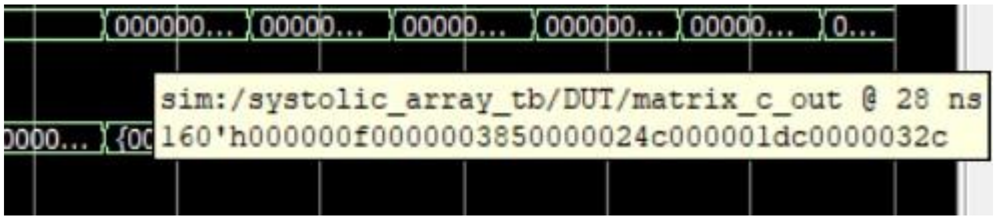
```
sim:/systolic_array_tb/DUT/matrix_elments @ 28 ns
0 : 32'h00000056 32'h0000027b 32'h000001fa 32'h000002d8 32'h00000305
1 : 32'h0000007d 32'h00000116 32'h000008b9 32'h000000c6 32'h000002c9
2 : 32'h000000fb 32'h00000343 32'h000002df 32'h000001d6 32'h0000028a
3 : 32'h0000009f 32'h0000037c 32'h000004cd 32'h000003d9 32'h00000533
4 : 32'h000000f0 32'h00000385 32'h0000024c 32'h000001dc 32'h0000032c
```

## First Row at Valid_out=1



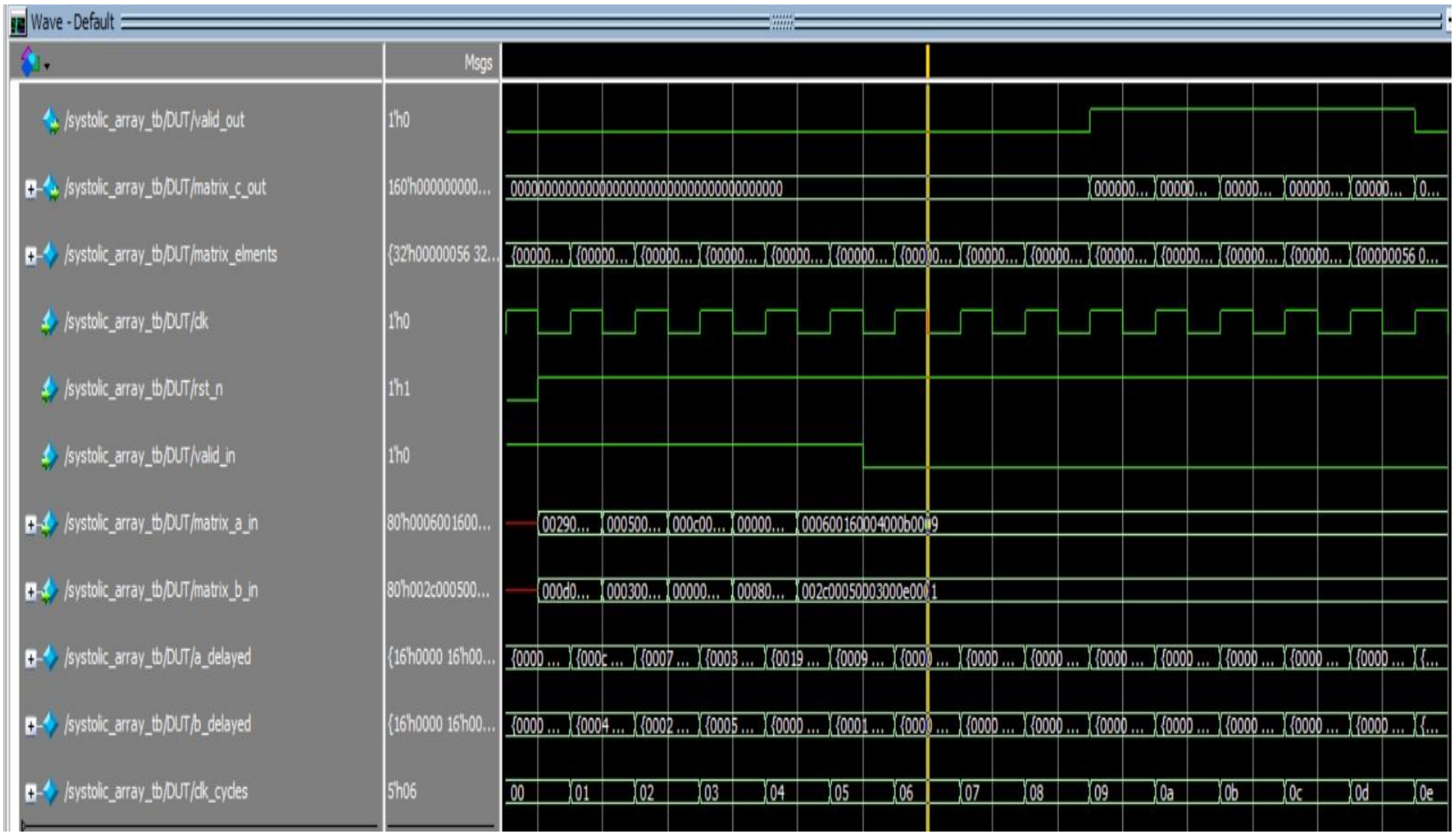
```
sim:/systolic_array_tb/DUT/matrix_c_out @ 19 ns
160'h000000560000027b000001fa000002d800000305
```

## Last Row at Valid_out=1



```
sim:/systolic_array_tb/DUT/matrix_c_out @ 28 ns
160'h000000f0000003850000024c000001dc0000032c
```

- **Complete Wave**



```
Wave - Default
                                          Msgs
/systolic_array_tb/DUT/valid_out          1'h0
/systolic_array_tb/DUT/matrix_c_out       160'h000000000...
/systolic_array_tb/DUT/matrix_elments     {32'h00000056 32...
/systolic_array_tb/DUT/clk                1'h0
/systolic_array_tb/DUT/rst_n              1'h1
/systolic_array_tb/DUT/valid_in           1'h0
/systolic_array_tb/DUT/matrix_a_in        80'h0006001600...
/systolic_array_tb/DUT/matrix_b_in        80'h002c000500...
/systolic_array_tb/DUT/a_delayed          {16'h0000 16'h00...
/systolic_array_tb/DUT/b_delayed          {16'h0000 16'h00...
/systolic_array_tb/DUT/clk_cycles         5'h06
```

000000000000000000000000000000000000000
000000... 00000... 00000... 000000... 00000... 0...
00290... 000500... 000c00... 00000... 000600160004000b0009
000d0... 000300... 00000... 00080... 002c00050003000e0001
00 01 02 03 04 05 06 07 08 09 0a 0b 0c 0d 0e