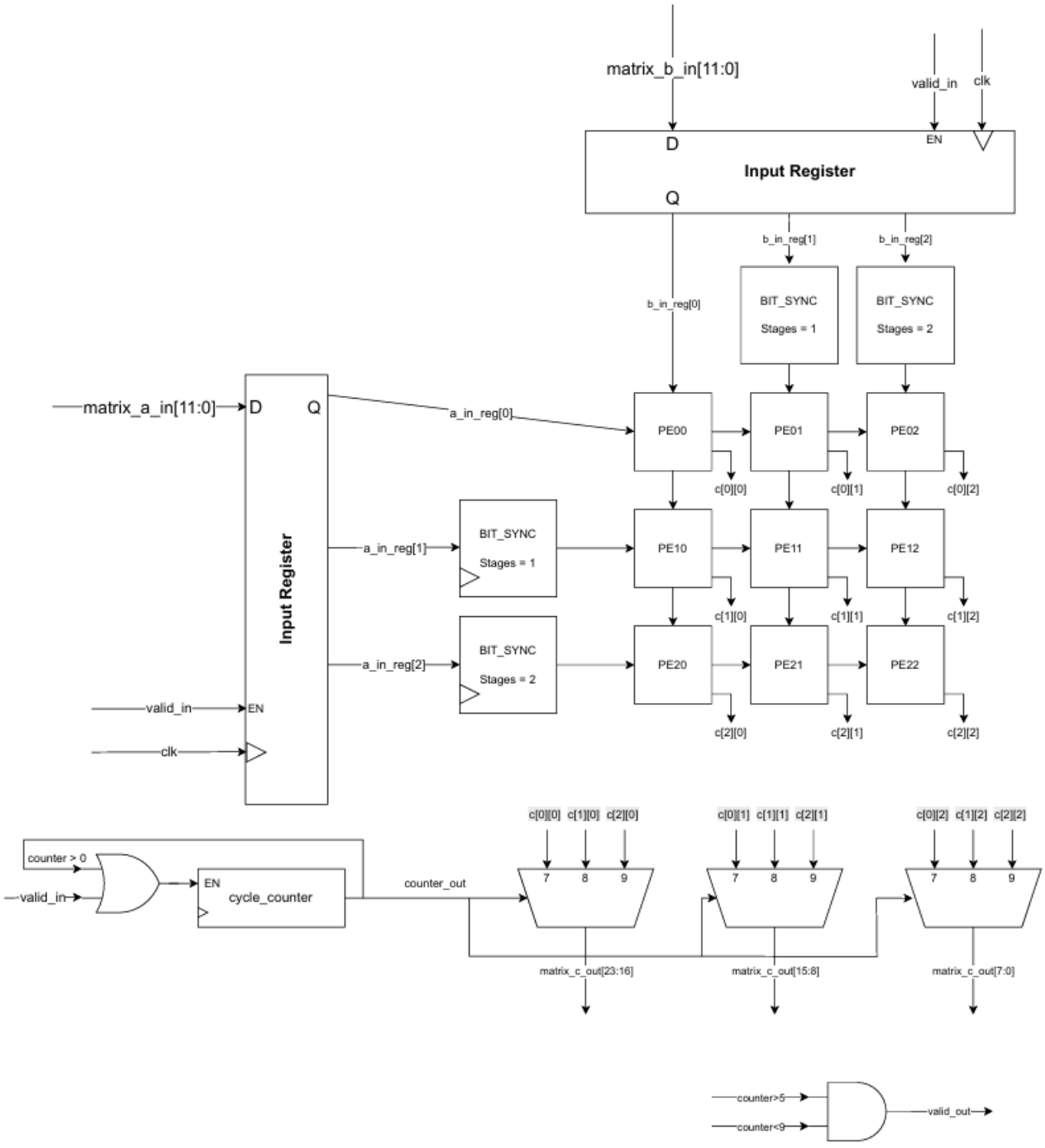
LAB 0 (Report)
Systolic Array for Matrix Multiplication

M.B. Kareem Atef Naguib

1.1 Architecture



1.2 Design Operation

- I have created the architecture above for N_SIZE = 3 and DATAWIDTH = 4, to explain and visualize the core idea of my design itself, the input data of matrix_a_in which represent the concatenation of each column elements together in one vector and for matrix_b_in which represent the concatenation of each row elements together in one vector and registered them using Input Register when enable signal valid_in asserted, otherwise it preserves its old value.

- Then I have created a block named BIT_SYNC which is simply a chain of flip flops their number defined by a parameter named NUM_STAGES to synchronize the flow of input matrices elements, for matrix_a_in side I used a BIT_SYNC with NUM_STAGES = 1 for the second row inputs same for the third row inputs but with NUM_STAGES = 2 and its generic my design will create 1 BIT_SYNC if N_SIZE = 2 and so on for other N_SIZE as 4 and 5, so it depends on the parameters values, the same was implemented for matrix_b_in side.

- For each PE it was simply a block that takes a_in and b_in and multiply and accumulate them in an internal signal in each PE which is c_out then pass a_in, b_in valus to the intended next PE.

- The output of the first row should be ready at cycle 4, but I added 2 idle cycles to ensure the output values are settled then take the first row output values at cycle 7, since design is pipelined, the second row output values is taken at cycle 8 and last one at cycle 9, that's explains the reasons for the muxes and the counter that counts the cycle, which is to assign right row outputs to matrix_c_out at the right time.

1.3 Code Description

a) PE

```
//////////////////////////////// 1) PE ////////////////////////////

module PE #(
    parameter DATAWIDTH = 4
)(
    input wire clk,
    input wire rst_n,
    input wire clear,
    input wire [DATAWIDTH-1:0] a_in,
    input wire [DATAWIDTH-1:0] b_in,
    output reg [DATAWIDTH-1:0] a_out,
    output reg [DATAWIDTH-1:0] b_out,
    output reg [2*DATAWIDTH-1:0] c_out
);

// Shows the simple function of the PEs which is mult. and acc. in the same PE

always @(posedge clk or negedge rst_n) begin
    if (!rst_n) begin
        a_out <= 0;
        b_out <= 0;
        c_out <= 0;
    end else if(!clear) begin
        a_out <= a_in;
        b_out <= b_in;
        c_out <= c_out + a_in * b_in;
    end
    else begin
        a_out <= 0;
        b_out <= 0;
        c_out <= 0;
    end
end
endmodule

//////////////////////////////////////////////////////////////////////////////
```

- This is the PE module which implements the multiplication and accumulation in the same PE block, and a clear signal is used to clear the PE for potential upcoming new values, and prevent from accumulating over old values.

b) BIT_SYNC

```
//////////////////////////////// 2) BIT_SYNC ////////////////////////////

module BIT_SYNC #(
    parameter BUS_WIDTH = 1,
              NUM_STAGES = 2
)

(
    input wire[BUS_WIDTH-1:0] ASYNC,
    input wire CLK,
    input wire RST,
    output reg[BUS_WIDTH-1:0] SYNC
);


reg[BUS_WIDTH-1:0] q[0:NUM_STAGES-1];
integer i;

always @(posedge CLK, negedge RST) begin
    if(!RST) begin
        for(i=0;i<NUM_STAGES;i=i+1)
        q[i]<=0;
    end
    else begin
        // First Flop accepts input signal
        q[0]<=ASYNC;
        // Sifting data between stages
        for(i=1;i<NUM_STAGES;i=i+1)
            q[i]<=q[i-1];

    end
end

// assigning last flop data to output
always @(*) SYNC=q[NUM_STAGES-1];

endmodule

//////////////////////////////////////////////////////////////////////
```

- I have designed this general block to add registers to the input side according to N_SIZE
  Ex: For N_SIZE = 2 -> generates 1 stage for second row of matrix A, same is applicable for columns of matrix B
  For N_SIZE = 3 -> generates 1 stage for second row of matrix A and 2 stages for third row, same is applicable for columns of matrix B

c) systolic_array (Top Module)

```
module systolic_array #(
    parameter DATAWIDTH = 4,
    parameter N_SIZE = 3
)(
    input wire clk,
    input wire rst_n,
    input wire valid_in,
    input wire [N_SIZE*DATAWIDTH-1:0] matrix_a_in,
    input wire [N_SIZE*DATAWIDTH-1:0] matrix_b_in,
    output reg valid_out,
    output reg [N_SIZE*2*DATAWIDTH-1:0] matrix_c_out
);

    //wires used for interconnections between PEs
    wire [DATAWIDTH-1:0] a[N_SIZE-1:0][N_SIZE-1:0];
    wire [DATAWIDTH-1:0] b[N_SIZE-1:0][N_SIZE-1:0];
    wire [2*DATAWIDTH-1:0] c[N_SIZE-1:0][N_SIZE-1:0];

    // Input registers that accepts matrix_a_in and matrix_b_in data
    reg [DATAWIDTH-1:0] a_in_reg[N_SIZE-1:0];
    reg [DATAWIDTH-1:0] b_in_reg[N_SIZE-1:0];

    // Output data from the regiesters for each column of matrix A (except first element) and for each row of B (except first element)
    wire[DATAWIDTH-1:0] a_in_sync[N_SIZE-2:0];
    wire[DATAWIDTH-1:0] b_in_sync[N_SIZE-2:0];

    reg clear_pe;
```

- Internal signals declarations with their descriptions

```
    integer i,j;

// Input processing
always @(posedge clk or negedge rst_n) begin
        if (!rst_n) begin
            for (i = 0; i < N_SIZE*2; i = i + 1) begin
                a_in_reg[i] <= 0;
                b_in_reg[i] <= 0;
            end
        end
else begin
            if (valid_in) begin
                for (i = 0; i < N_SIZE; i = i + 1) begin
                    // extracts each element from matrix_a_in and assign it to seperate var a_in_reg starts from least to most
                    // and same for matrix_b_in
                    // Ex: a_in_reg[0] = matrix_a_in[0 +: 4] -> bits [3:0]
                    a_in_reg[i] <= matrix_a_in[i*DATAWIDTH +: DATAWIDTH];
                    b_in_reg[i] <= matrix_b_in[i*DATAWIDTH +: DATAWIDTH];
                end
            end
            else begin
                // perserving the old value for valid_in = 0
              for (i = 0; i < N_SIZE; i = i + 1) begin
                a_in_reg[i] <= a_in_reg[i];
                a_in_reg[i] <= a_in_reg[i];
            end
            end

    end
    end
```

- Input processing, in which when valid_in asserted extract each element from matrix_a_in and assign it to a_in_reg and same for matrix_b_in.
- Otherwise, it preserves its value

```
genvar k,m;
generate
    for(k=0;k<N_SIZE-1;k=k+1) begin
        BIT_SYNC #(.BUS_WIDTH(DATAWIDTH),.NUM_STAGES(k+1)) sync_inst(
            .ASYNC(a_in_reg[k+1]),
            .CLK(clk),
            .RST(rst),
            .SYNC(a_in_sync[k])
        );
    end
    for(k=0;k<N_SIZE-1;k=k+1) begin
        BIT_SYNC #(.BUS_WIDTH(DATAWIDTH),.NUM_STAGES(k+1)) sync_inst(
            .ASYNC(b_in_reg[k+1]),
            .CLK(clk),
            .RST(rst),
            .SYNC(b_in_sync[k])
        );
    end
     endgenerate
```

- generate block used to create the instantiations of BIT_SYNC (regs) for A and B according to N_SIZE

```
    // Generate PE array instantiations
 genvar col, row;
generate
    for (row = 0; row < N_SIZE; row = row + 1) begin
        for (col = 0; col < N_SIZE; col = col + 1) begin
            wire [DATAWIDTH-1:0] a_in_wire, b_in_wire;

                // first element accepts value from input regiester directly
            if (col == 0 && row ==0)
                assign a_in_wire = a_in_reg[row];

                // for other rows and first col accepts data coming from the BIT_SYNC (regs) outputs
            else if (col ==0 && row !=0)
                assign a_in_wire = a_in_sync[row-1];

                // otherwise accepts the internal wire from the previous PE
            else
                assign a_in_wire = a[row][col-1];

                // first element accepts value from input regiester directly
            if (row == 0 && col == 0)
                assign b_in_wire = b_in_reg[col];

                // for other rows and first col accepts data coming from the BIT_SYNC (regs) outputs
            else if(row == 0 && col != 0)
                assign b_in_wire = b_in_sync[col-1];

                // otherwise accepts the internal wire from the previous PE
            else
                assign b_in_wire = b[row-1][col];

            PE #(.DATAWIDTH(DATAWIDTH)) pe_inst (
                .clk(clk),
                .rst_n(rst_n),
                .clear(clear_pe),
                .a_in(a_in_wire),
                .b_in(b_in_wire),
                .a_out(a[row][col]),
                .b_out(b[row][col]),
                .c_out(c[row][col])
            );
        end
    end
endgenerate
```

- Generation of PEs array with its description commented.

```
// Generalized control logic
reg [31:0] cycle_count;
integer row_idx;

always @(posedge clk or negedge rst_n) begin
    if (!rst_n) begin
        cycle_count <= 0;
        valid_out <= 0;
        matrix_c_out <= 0;
    end else begin
        if (cycle_count == (3 * N_SIZE + 1)) begin
         cycle_count <= 0;
         clear_pe <= 1;
        end
        // Count cycles after first valid input
        else if (valid_in || cycle_count > 0) begin
            clear_pe <= 0;
            cycle_count <= cycle_count + 1;
        end

        // Generate valid output at correct times
        if (cycle_count >= 2*N_SIZE && cycle_count < 3*N_SIZE) begin
            valid_out <= 1;

            // for each valid output's cycle count we start assigning each element in each row for matrix_c_out
            row_idx = cycle_count - (2*N_SIZE);
            if (row_idx < N_SIZE) begin
                for (i = N_SIZE - 1; i >= 0; i = i - 1) begin
                    matrix_c_out[(N_SIZE - 1 - i) * 2 * DATAWIDTH +: 2 * DATAWIDTH] <= c[row_idx][i];
                end
            end else begin
                matrix_c_out <= 0;
            end
        end else begin
            valid_out <= 0;
        end
    end

end

endmodule
```

- Control logic starts with cycle_count starts incrementing as soon as the first valid input is received then keeps tracking the processing cycles regarding valid_in.
- we start receiving rows of the result matrix when cycle_count reaches 2*N_SIZE and releases output of each row every cycle.
- Then we determine which row from the result matrix should be the output, and packs the row values into a flattened matrix_c_out bus.
- valid_out is kept asserted during this period of result window.
- Finally assert the clear signal to clear all PEs and cycle_count after ensuring that the output was released, and that's to prevent accumulating if there were new values coming.

2. Simulation Snaps

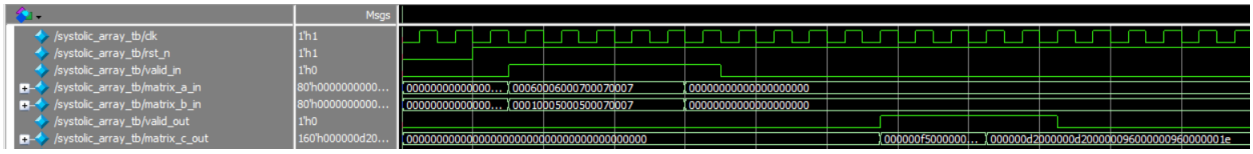
- I have implemented 6 test cases (DATAWIDTH is kept 16)

  a) Default Case for N_SIZE = 5:

```
matrix A = [7  7  7  7  7]
           7  7  7  7  7
           7  7  7  7  7
           6  6  6  6  6
           6  6  6  6  6

matrix B = [7  7  5  5  1]
           7  7  5  5  1
           7  7  5  5  1
           7  7  5  5  1
           7  7  5  5  1

matrix C = [245, 245, 175, 175,  35]
           [245, 245, 175, 175,  35]
           [245, 245, 175, 175,  35]
           [210, 210, 150, 150,  30]
           [210, 210, 150, 150,  30]
```




```
# ****INPUT DATA****
# First Column in A : 7 , 7, 7, 6, 6
# First Row in B : 7 , 7, 5, 5, 1
# Second Column in A : 7 , 7, 7, 6, 6
# Second Row in B : 7 , 7, 5, 5, 1
# Third Column in A : 7 , 7, 7, 6, 6
# Third Row in B : 7 , 7, 5, 5, 1
# Fourth Column in A : 7 , 7, 7, 6, 6
# Fourth Row in B : 7 , 7, 5, 5, 1
# Fifth Column in A : 7 , 7, 7, 6, 6
# Fifth Row in B : 7 , 7, 5, 5, 1
# Output row: 245 245 175 175 35
# Output row: 245 245 175 175 35
# Output row: 245 245 175 175 35
# Output row: 210 210 150 150 30
# Output row: 210 210 150 150 30
```

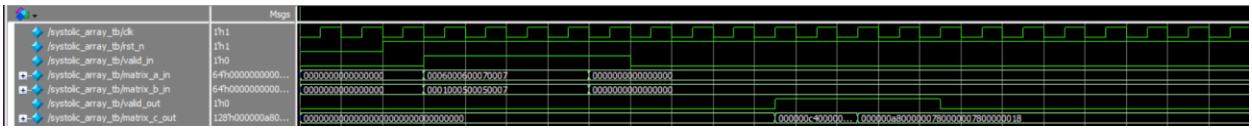
- The log showing the design is working as expected.

b) Case for N_SIZE = 4:

```
matrix A = [7  7  7  7 ]
           7  7  7  7
           6  6  6  6
           6  6  6  6

matrix B = [7  5  5 1 ]
           7  5  5 1
           7  5  5 1
           7  5  5 1

matrix C = [196, 140, 140, 28]
           [196, 140, 140, 28]
           [168, 120, 120, 24]
           [168, 120, 120, 24]
```



```
# ****INPUT DATA****
# First Column in A : 7 , 7, 6, 6
# First Row in B : 7 , 5, 5, 1
# Second Column in A : 7 , 7, 6, 6
# Second Row in B : 7 , 5, 5, 1
# Third Column in A : 7 , 7, 6, 6
# Third Row in B : 7 , 5, 5, 1
# Fourth Column in A : 7 , 7, 6, 6
# Fourth Row in B : 7 , 5, 5, 1
# Output row: 196 140 140 28
# Output row: 196 140 140 28
# Output row: 168 120 120 24
# Output row: 168 120 120 24
```

- The log showing the design is working as expected.

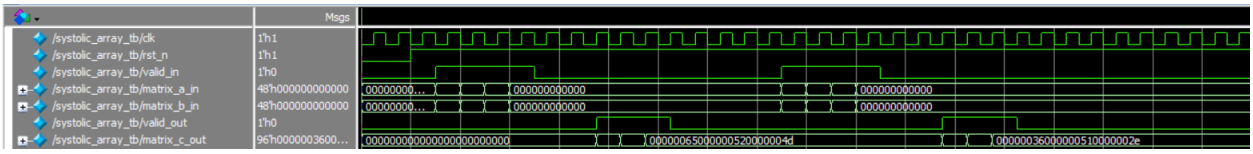
c) 2 Test Cases for N_SIZE = 3:

$$A = \begin{bmatrix} 1 & 4 & 8 \\ 9 & 2 & 5 \\ 6 & 7 & 3 \end{bmatrix}, \quad B = \begin{bmatrix} 4 & 5 & 9 \\ 8 & 7 & 2 \\ 7 & 1 & 3 \end{bmatrix}$$

$$C = A \times B = \begin{bmatrix} 92 & 41 & 41 \\ 87 & 64 & 100 \\ 101 & 82 & 77 \end{bmatrix}$$

$$A = \begin{bmatrix} 7 & 4 & 8 \\ 6 & 2 & 4 \\ 6 & 8 & 1 \end{bmatrix}, \quad B = \begin{bmatrix} 5 & 5 & 1 \\ 2 & 6 & 4 \\ 8 & 3 & 8 \end{bmatrix}$$

$$C = A \times B = \begin{bmatrix} 107 & 83 & 87 \\ 66 & 54 & 46 \\ 54 & 81 & 46 \end{bmatrix}$$




```
# ****INPUT DATA****
# First Column in A : 1 , 9, 6
# First Row in B : 4 , 5, 9
# Second Column in A : 4 , 2, 7
# Second Row in B : 8 , 7, 2
# Third Column in A : 8 , 5, 3
# Third Row in B : 7 , 1, 3
# Output row: 92 41 41
# Output row: 87 64 100
# Output row: 101 82 77
# First Column in A : 7 , 6, 6
# First Row in B : 5 , 5, 1
# Second Column in A : 4 , 2, 8
# Second Row in B : 2 , 6, 4
# Third Column in A : 8 , 4, 1
# Third Row in B : 8 , 3, 8
# Output row: 107 83 87
# Output row: 66 54 46
# Output row: 54 81 46
```

- The log showing the design is working as expected.

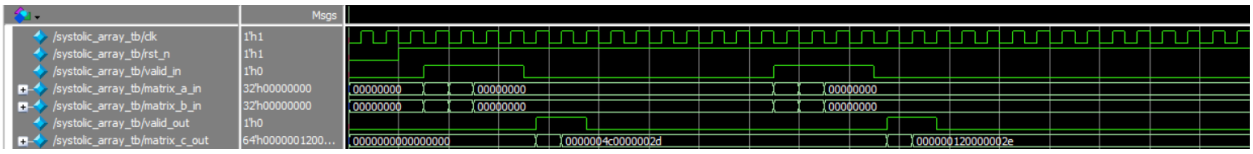
d) 2 Test Cases for N_SIZE = 2:

$$A = \begin{bmatrix} 2 & 1 \\ 5 & 7 \end{bmatrix}, \qquad B = \begin{bmatrix} 4 & 2 \\ 8 & 5 \end{bmatrix}$$

$$C = A \times B = \begin{bmatrix} 16 & 9 \\ 76 & 45 \end{bmatrix}$$

$$A = \begin{bmatrix} 7 & 9 \\ 2 & 6 \end{bmatrix}, \qquad B = \begin{bmatrix} 3 & 8 \\ 2 & 5 \end{bmatrix}$$

$$C = A \times B = \begin{bmatrix} 39 & 101 \\ 18 & 46 \end{bmatrix}$$




```
# ****INPUT DATA****
# First Column in A : 2 , 5
# First Row in B : 4 , 2
# Second Column in A : 1 , 7
# Second Row in B : 8 , 5
# Output row: 16 9
# Output row: 76 45
# First Column in A : 7 , 2
# First Row in B : 3 , 8
# Second Column in A : 9 , 6
# Second Row in B : 2 , 5
# Output row: 39 101
# Output row: 18 46
```

- The log showing the design is working as expected.