Google Tensor Processing Unit

- Google getting into hardware design. Three spins of TPU already!
- TPU is a specialized chip for AI $\rightarrow$ matrix-matrix/tensor operations
- **Key Idea**: Systolic design
  - Lots of simple parallel computational units
  - Data movement strictly localized to immediate neighbours
- Tightly coupled with TensorFlow (programming APIs)
- \$6.50 USD per TPU per hour (as of today!)

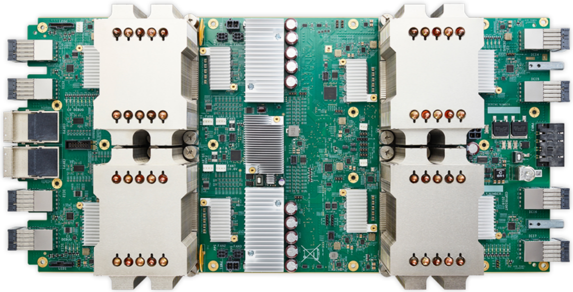


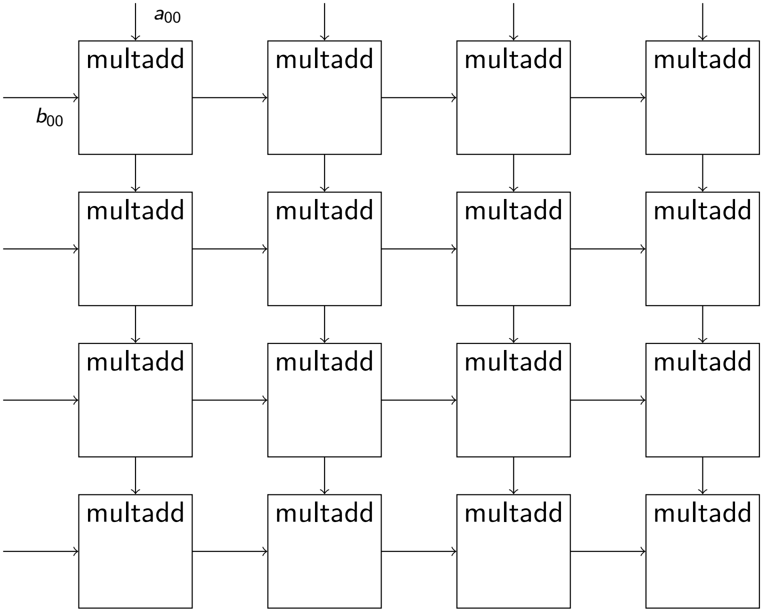
Figure: TPU v3 device → liquid cooled!

https://cloud.google.com/tpu/

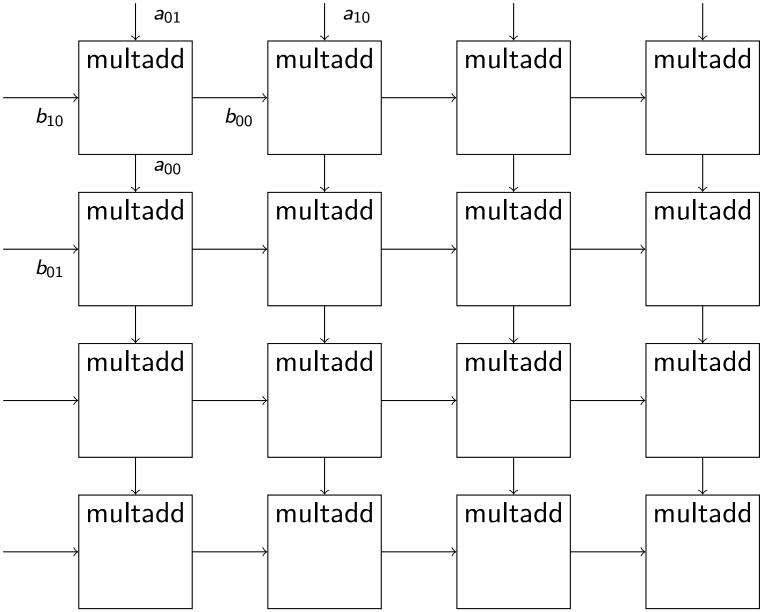
TPU History

- First generation
  - 64K 8-bit multiply-add operations @700 MHz $\rightarrow$ 44 Tops/s
- Second generation
  - 32K 32-bit (and single-precision float) multiply-add operations @700 MHz $\rightarrow$ 45 TFLOPS (float)
- Third generation
  - Not a lot of details $\rightarrow$ 2$\times$ better than TPU v2

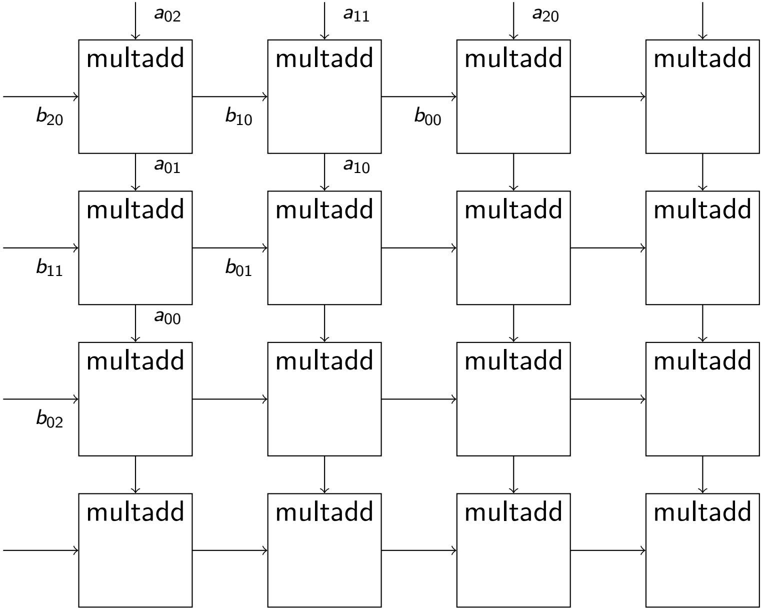
Systolic Array (Step-by-step Matrix Multiply)



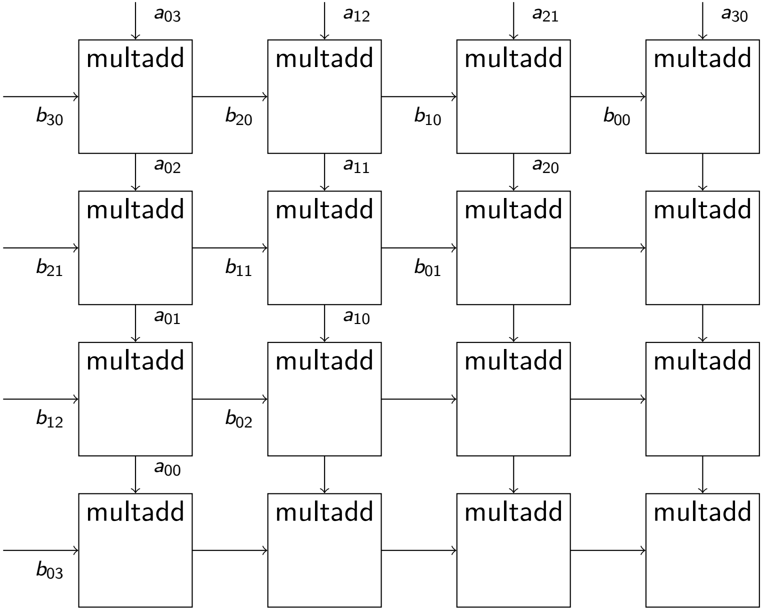
Systolic Array (Step-by-step Matrix Multiply)



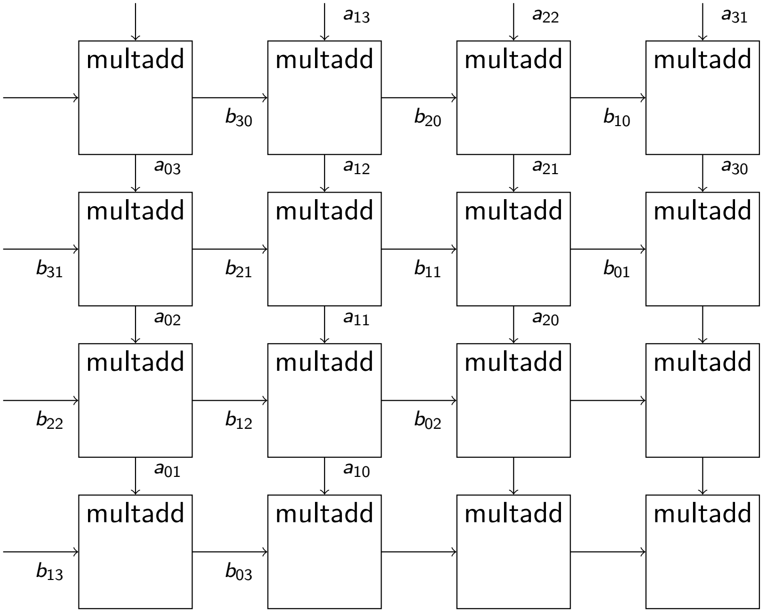
Systolic Array (Step-by-step Matrix Multiply)



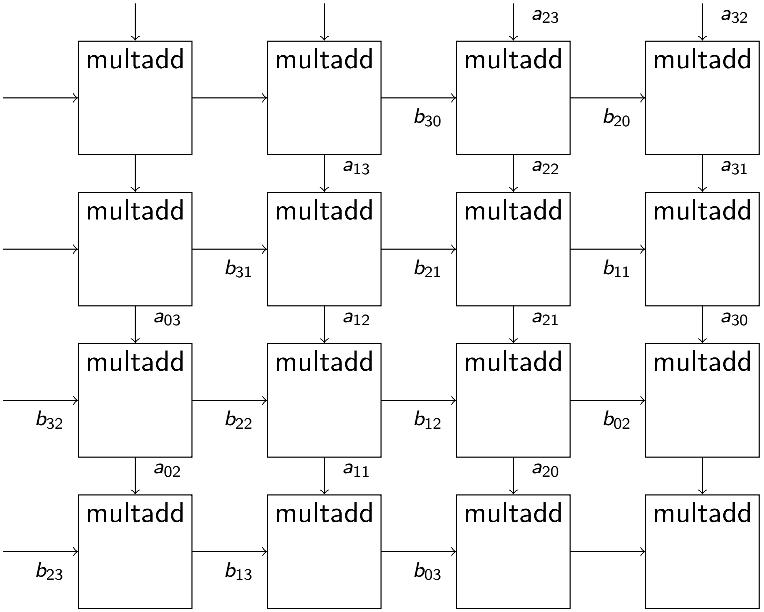
Systolic Array (Step-by-step Matrix Multiply)



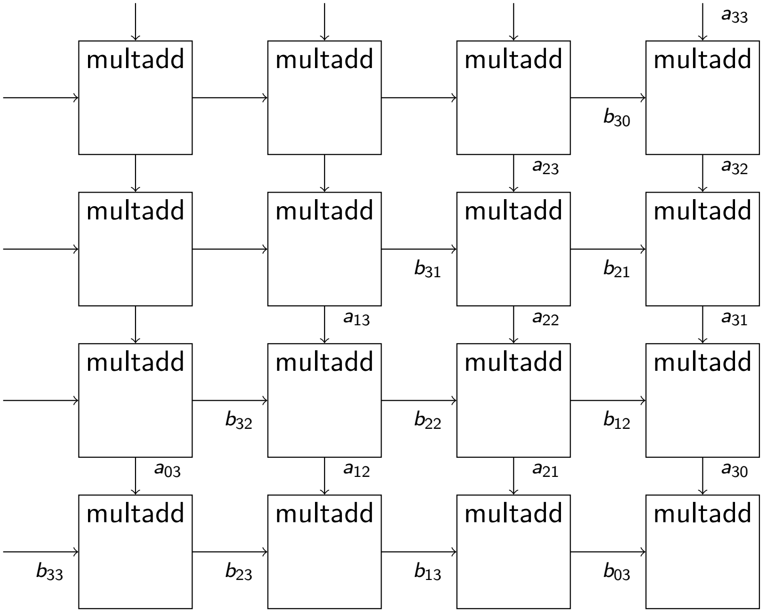
Systolic Array (Step-by-step Matrix Multiply)



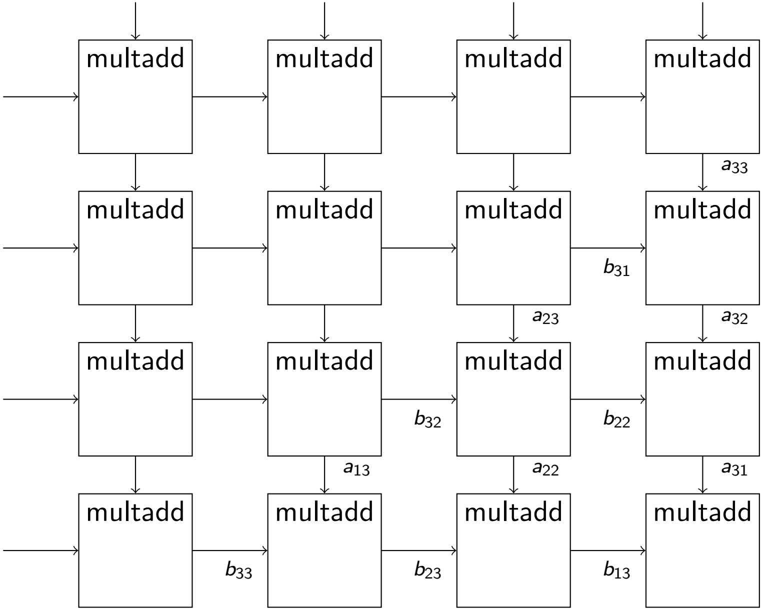
Systolic Array (Step-by-step Matrix Multiply)



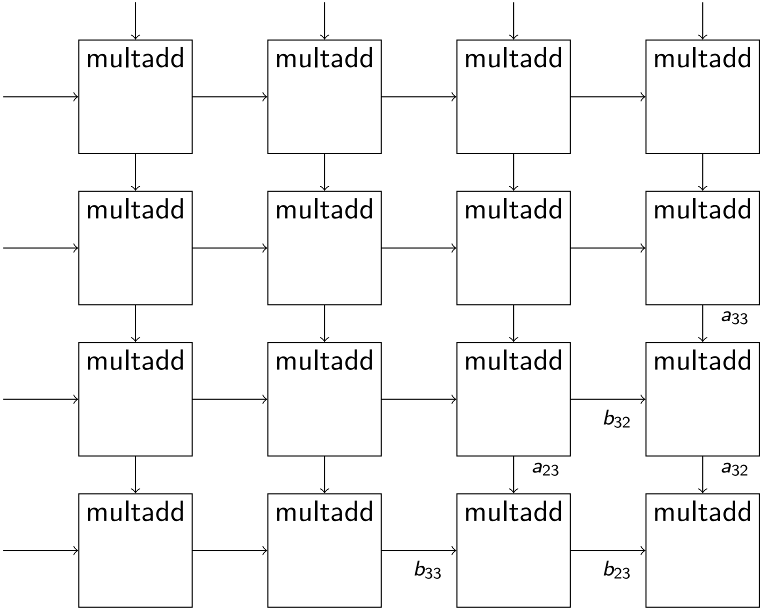
Systolic Array (Step-by-step Matrix Multiply)



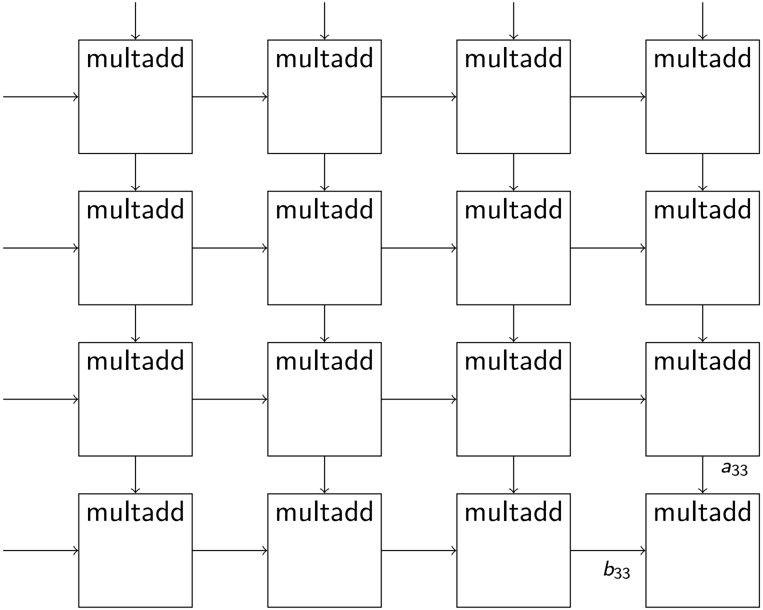
Systolic Array (Step-by-step Matrix Multiply)



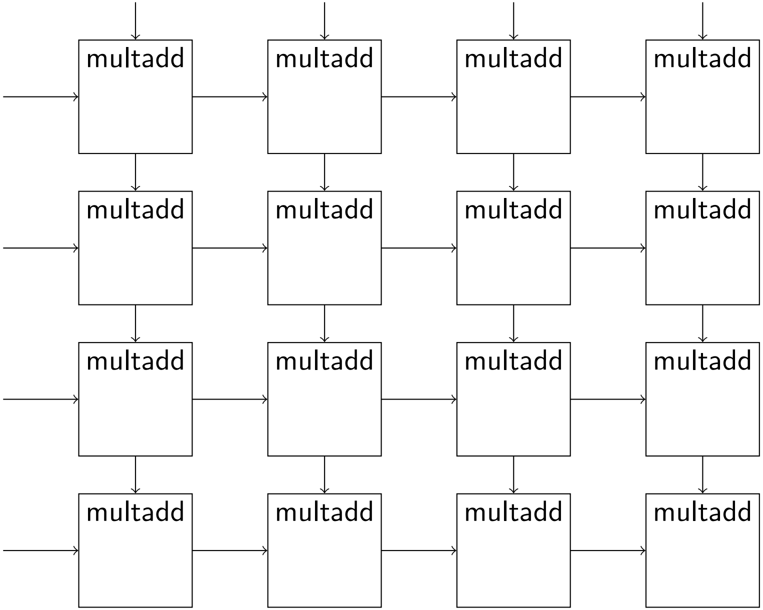
Systolic Array (Step-by-step Matrix Multiply)



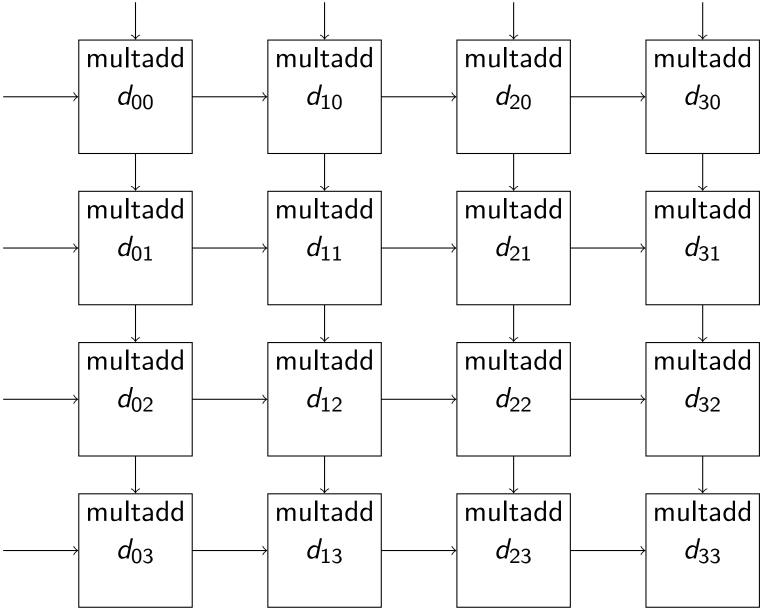
Systolic Array (Step-by-step Matrix Multiply)



Systolic Array (Step-by-step Matrix Multiply)



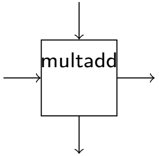
Systolic Array (Step-by-step Matrix Multiply)

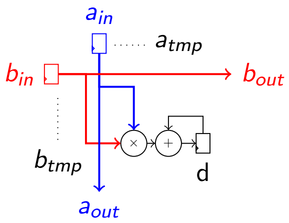


Matrix-Matrix Multiplication on Systolic Array

- Send A matrix along column, and B matrix along rows of a 2D array
- Each step along the way, the compute block performs a small multiply-add operation
- Data movement is co-ordinated to ensure correct data arrives at correct compute block in the correct time!
- All data movement is local $\rightarrow$ hardware design is super easy!
- In the end, final product D is available in every compute block $\rightarrow$ result must be read out (not shown)

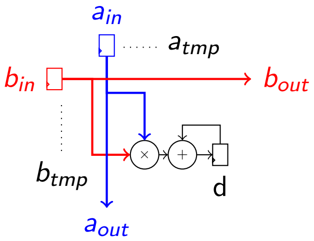
The `multadd` Building Block





- The `multadd` block is a simple 8b×8b multiplier followed by a 32b accumulator
- All communication in the chip is strictly local → nearest neighbour → no long wires!
  - Matrix elements *streamed* into systolic core from top+left links
  - Data forwarded to links below+right for further use by neighouring elements
- The building block in the TPUv1 was just 8b multiplication of neuron weights, and input → later revisions added floating-point and custom formats
- Design is so easy, even Google engineers can build this :)

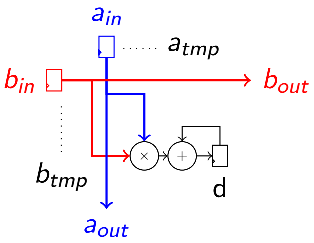
RTL Description of the `multadd` block



TPUv1 has 8b multipliers, 32b accumulators

```
module systolic_leaf
    (input clk,
     input rst,
     input [7:0] a_in,
     input [7:0] b_in,
     output wire [7:0] a_out,
     output wire [7:0] b_out
     );
```

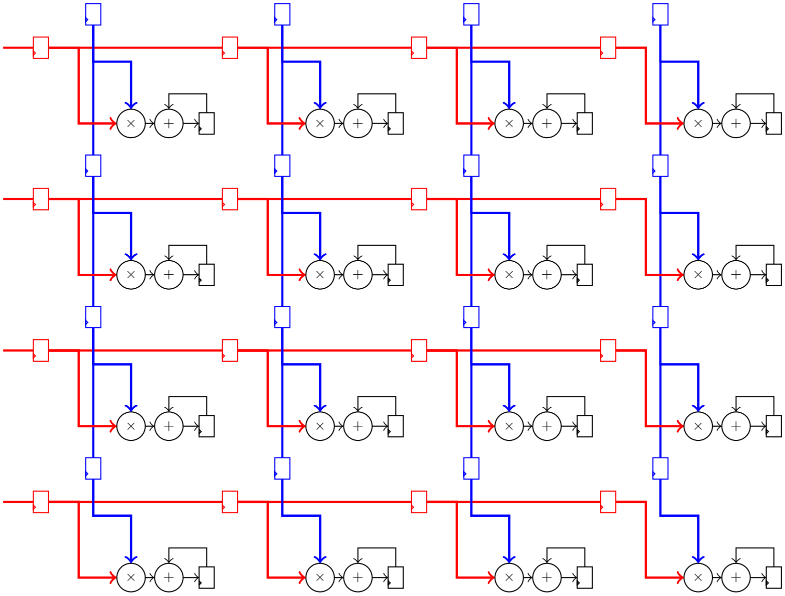
RTL Description of the multadd block



TPUv1 has 8b multipliers, 32b accumulators

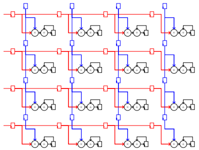
```
reg [31:0] d;
reg [7:0] a_tmp;
reg [7:0] b_tmp;

always@(posedge clk) begin
  if (rst) begin
    d     <= 0;
    a_tmp <= 0;
    b_tmp <= 0;
  end else begin
    d <= d + a_tmp*b_tmp;
    a_tmp <= a_in;
    b_tmp <= b_in;
  end
end

assign a_out = a_tmp;
assign b_out = b_tmp;

endmodule
```

Systolic Array Design (4×4 design)



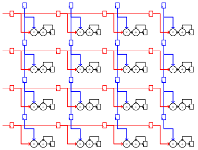
Caveat: Above pic is simple, actual TPUv1 is 256×256 blocks

RTL Description of the Systolic Array



```
module systolic_complete
  # (parameter SIZE = 4)
    (input clk,
     input rst,
     input wire [7:0] a [SIZE-1:0],
     input wire [7:0] b [SIZE-1:0]
     );
```

RTL Description of the Systolic Array



```
genvar i,j;

wire [7:0] horizontal_wires [SIZE+1][SIZE];
wire [7:0] vertical_wires [SIZE][SIZE+1];

generate
for (i=0; i<=SIZE-1; i=i+1) begin : row
  assign vertical_wires[0][i] = b[i];
  assign horizontal_wires[i][0] = a[i];
  for (j=0; j<=SIZE-1; j=j+1) begin : col
    systolic_leaf
    systolic_leaf_inst (
      .clk   (clk),
      .rst   (rst),
      .a_in  (vertical_wires[i][j]),
      .b_in  (horizontal_wires[i][j]),
      .a_out (vertical_wires[i+1][j]),
      .b_out (horizontal_wires[i][j+1])
      );
    end
  end
endgenerate

endmodule
```