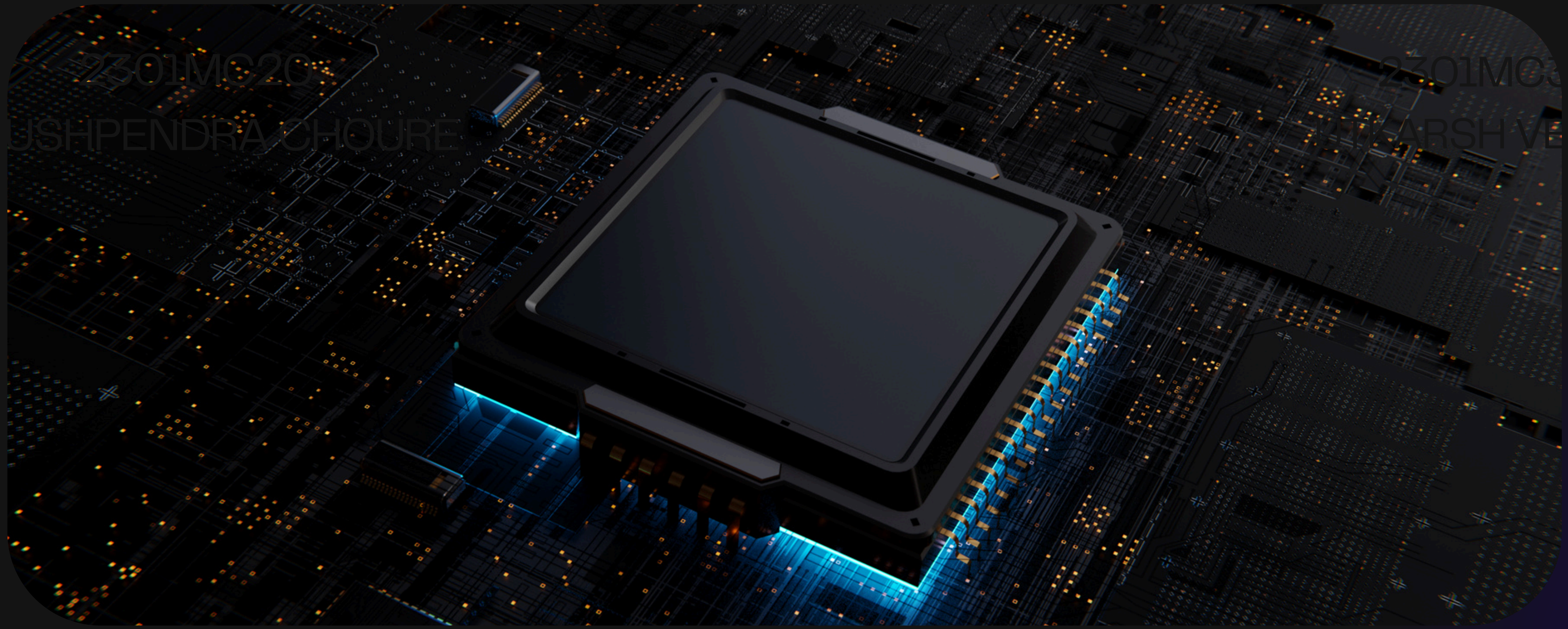
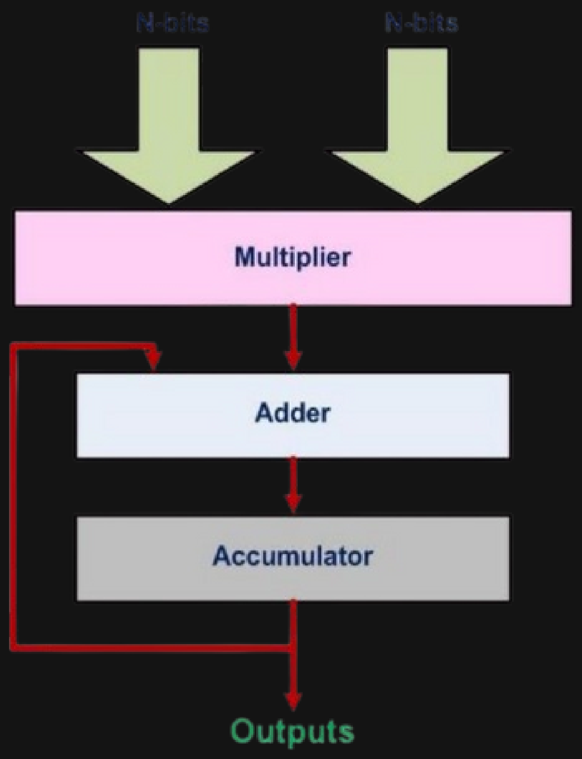
Tensor Processing Unit


2301MC20
USHPENDRA CHOURE
2301MC3
KARSH VE

What is a TPU



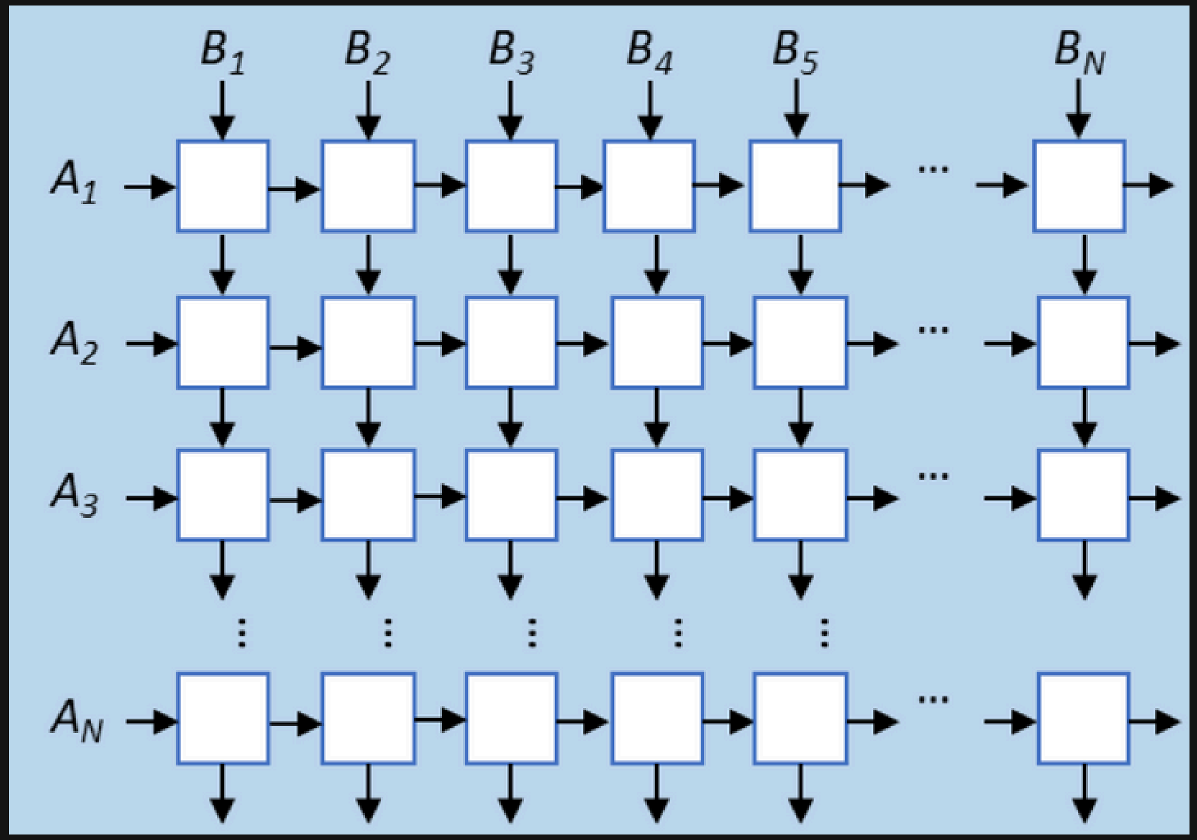
- Deep learning models involve millions of matrix and tensor operations
- Traditional CPUs and GPUs are general-purpose processors, which are not always efficient.
- A Tensor Processing Unit (TPU) specialised hardware for accelerating machine learning tasks, particularly matrix multiplications in deep neural networks
- Originally developed by Google

Processing Elements(PE)

- Processing elements (PE) are the basic units of the TPU
- A processing element is an ALU which performs two operations:
  - Multiply two numbers
  - Add the result to an accumulator (which holds a running total)

Systolic Array

- A systolic array is a grid of PEs that rhythmically compute and pass data through the array
- Google's TPU v1 features a 256x256 systolic array, totaling 65,536 ALUs



CPU vs GPU vs TPU

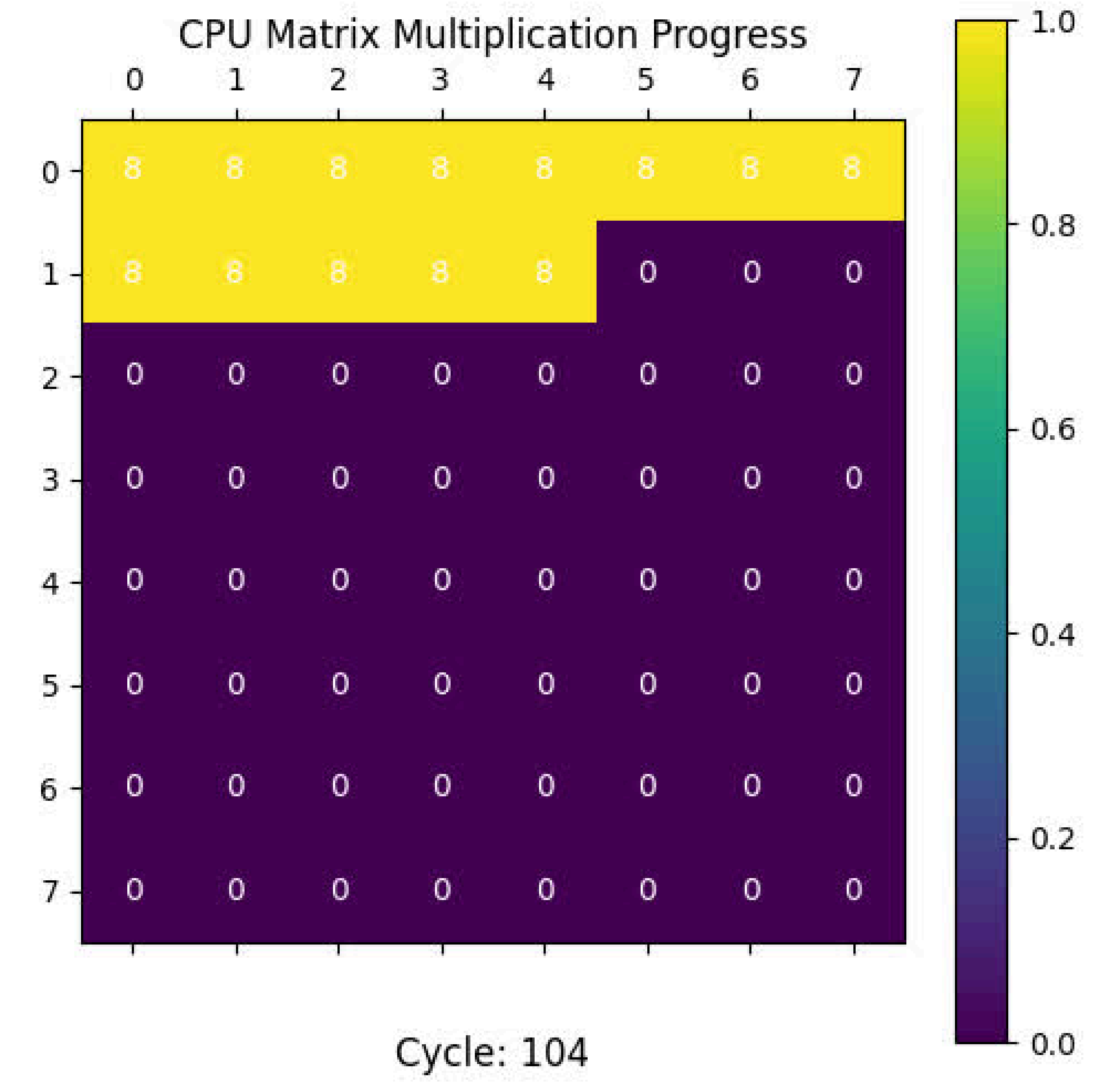
From the lens of tensor multiplication

CPU

CPUs are general-purpose processors designed for handling a wide range of tasks, from everyday computing (web browsing, office work) to gaming and multitasking. They excel at managing complex workflows with low latency, and running operating systems efficiently.

Example

- Memory Access:
  - ~1536 global memory accesses
  - Type: Caching, registers (on-chip)
- Cores:
  - Number: Single core
  - Type: High-performance general-purpose cores
- Parallelism: Limited (few threads per core)
- Purpose: Versatile computing



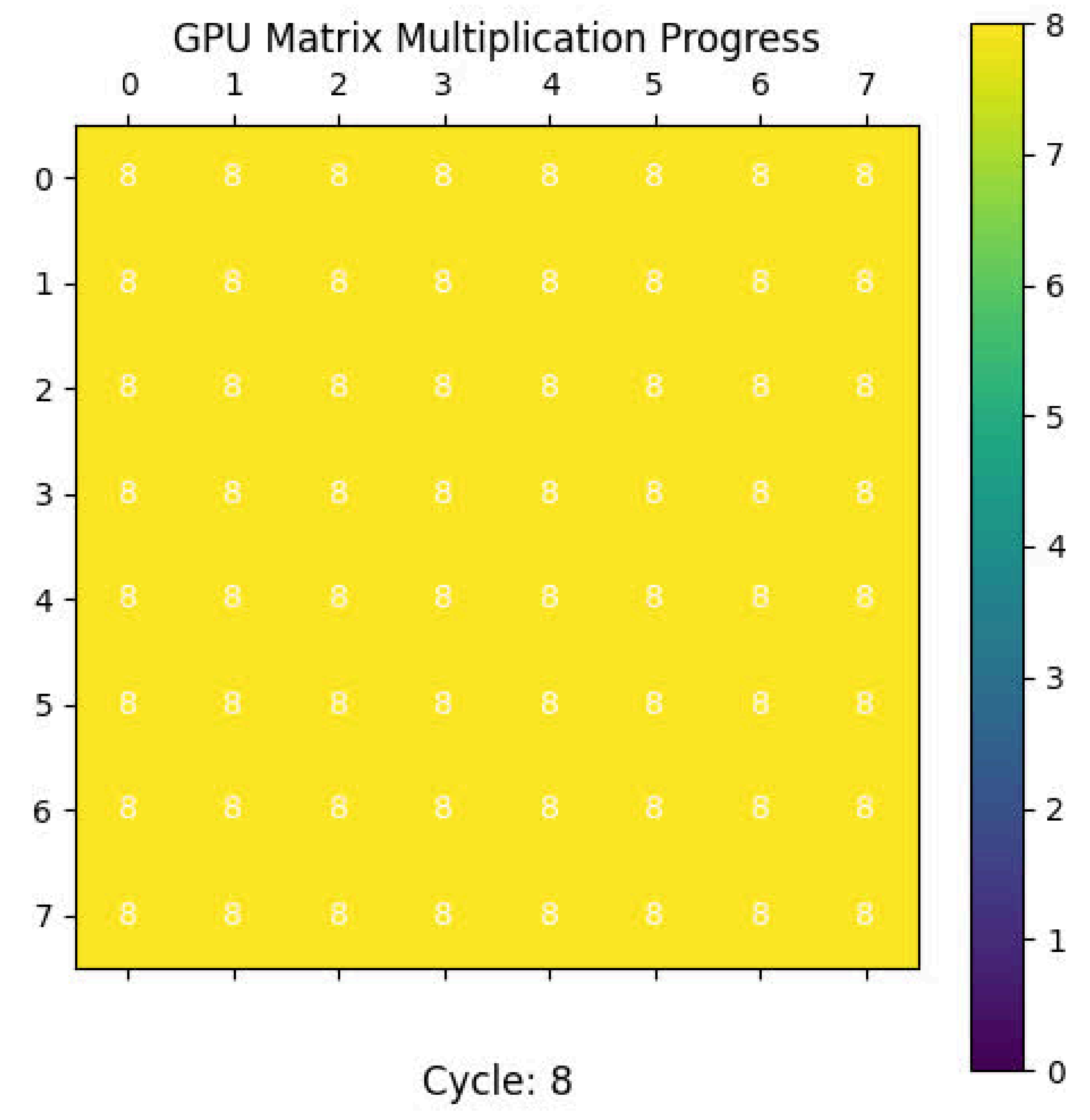
	0	1	2	3	4	5	6	7
0	8	8	8	8	8	8	8	8
1	8	8	8	8	8	0	0	0
2	0	0	0	0	0	0	0	0
3	0	0	0	0	0	0	0	0
4	0	0	0	0	0	0	0	0
5	0	0	0	0	0	0	0	0
6	0	0	0	0	0	0	0	0
7	0	0	0	0	0	0	0	0

GPU

GPUs are specialized for parallel processing, making them ideal for graphics rendering, AI training, and scientific computations. With thousands of cores, they accelerate tasks that require massive data parallelism, such as gaming, deep learning, and simulations.

Example

- Memory Access:
  - ~1088 global memory calls
  - Type: Shared memory (on-chip), coalescing
- Cores:
  - Number: 64 cores
  - Type: Parallel cores for large-scale parallelism
- Parallelism: Limited (few threads per core)
- Purpose: Versatile computing



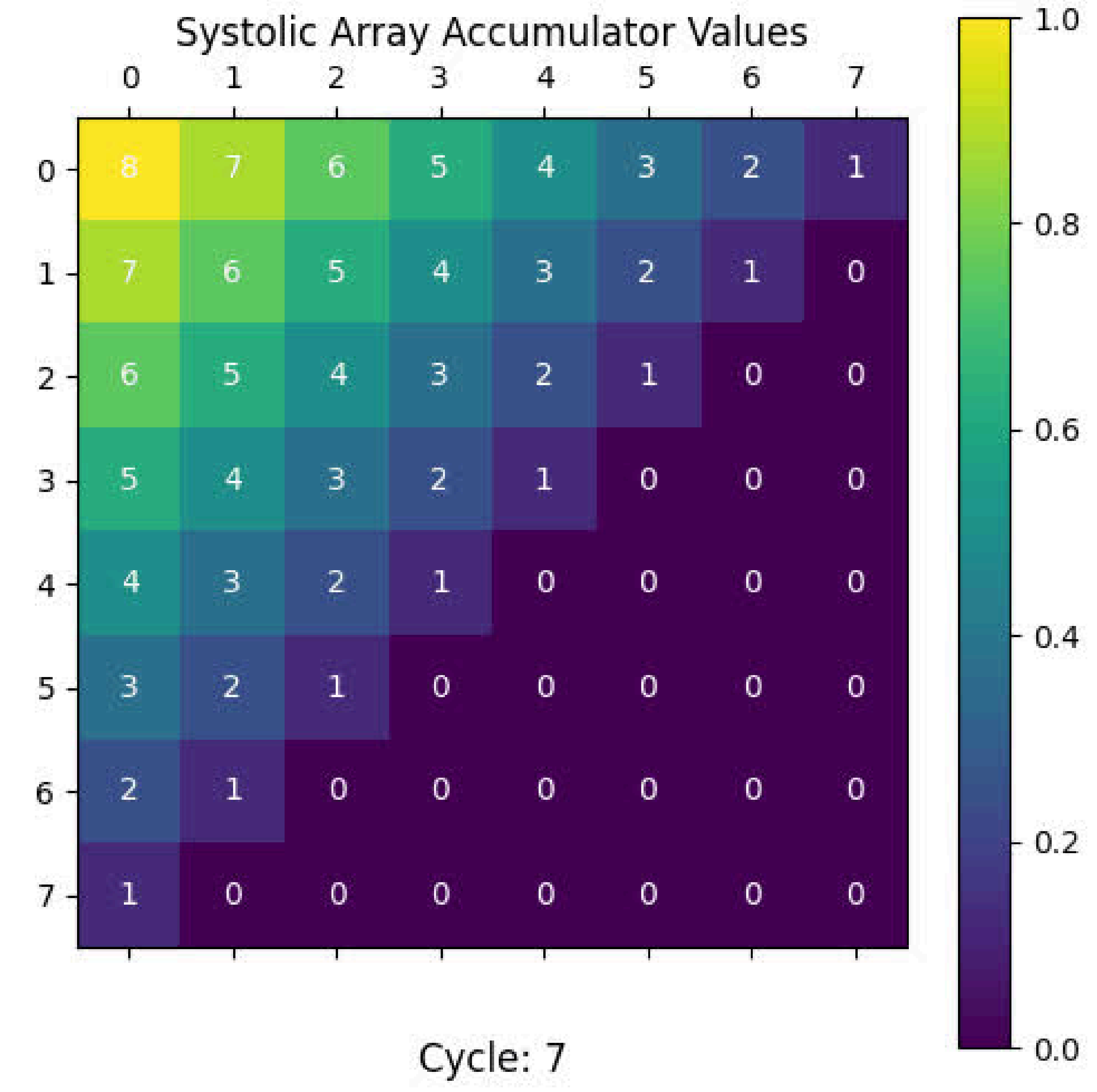
	0	1	2	3	4	5	6	7
0	8	8	8	8	8	8	8	8
1	8	8	8	8	8	8	8	8
2	8	8	8	8	8	8	8	8
3	8	8	8	8	8	8	8	8
4	8	8	8	8	8	8	8	8
5	8	8	8	8	8	8	8	8
6	8	8	8	8	8	8	8	8
7	8	8	8	8	8	8	8	8

TPU

TPUs are custom-built for machine learning workloads, optimizing tensor operations (matrix multiplications) at ultra-high speeds. They are used primarily in AI inference and training, reducing power consumption and latency compared to CPUs and GPUs for large-scale neural networks.

Example

- Memory Access:
  - ~192 global calls (128 input, 64 output)
  - Type: Systolic array, data streaming
- Cores:
  - Number: 64 PEs
  - Type: Specialized for tensor operations
- Parallelism: Massive, fine-tuned for tensor operations
- Purpose: Machine learning and tensor processing



Accelerating Matrix Multiplication with a Custom TPU Design

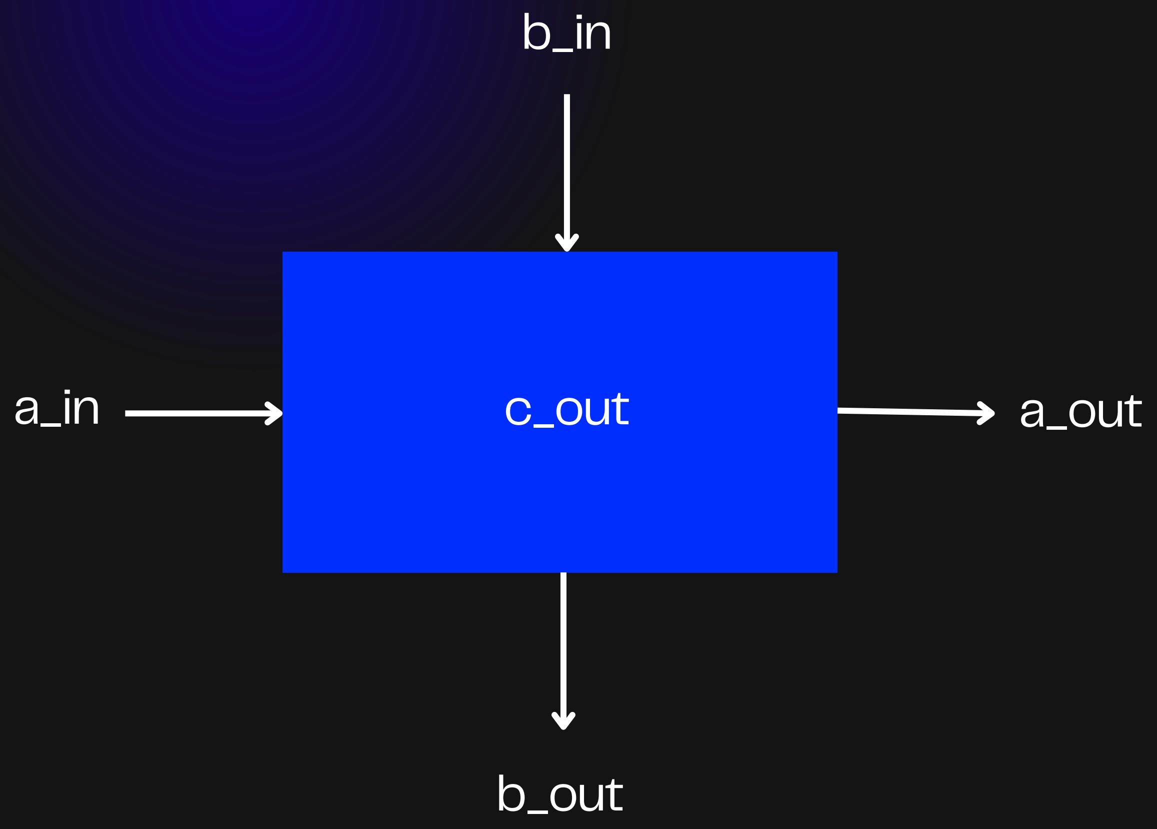
To implement the matrix multiply accelerator, the design is divided into modular Verilog components.
The hierarchy includes:

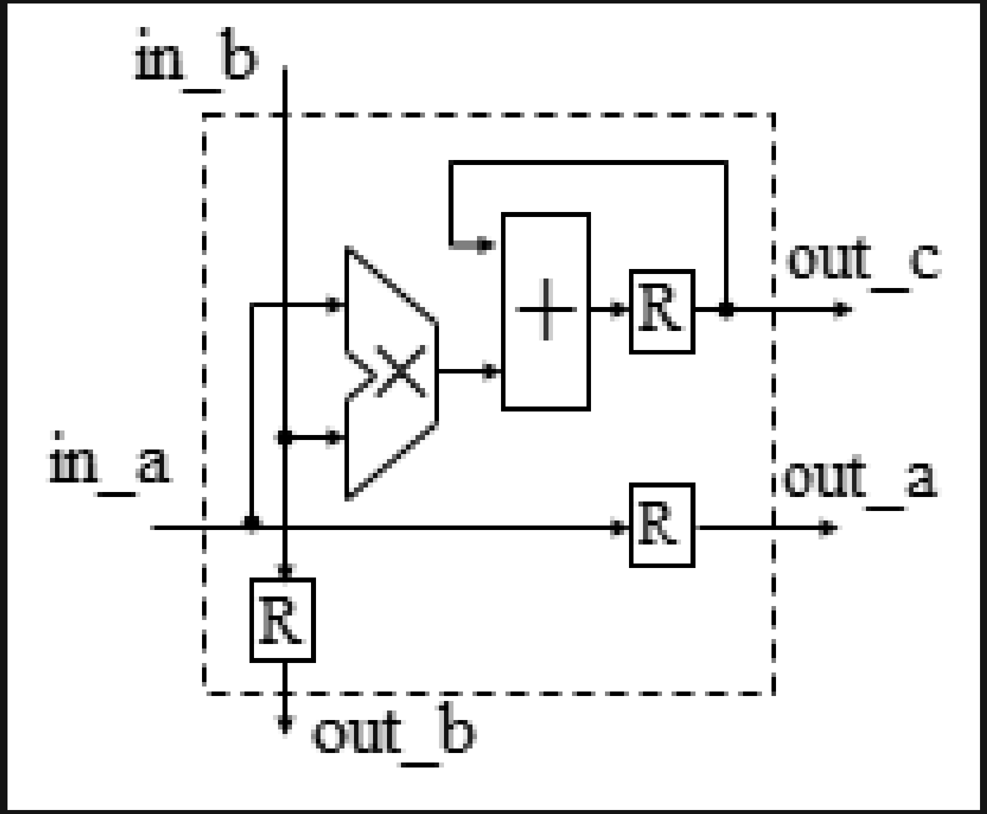
1. the Processing Element (PE), which is the basic compute node
2. the SystolicArray8x8, which instantiates a 8×8 mesh of PEs and handles data movement between them
3. the MatMul8x8_Top top-level module, which feeds input matrices into the array and collects the results
4. the tb_MatMul8x8 testbench, which provides stimulus (the matrices to multiply)

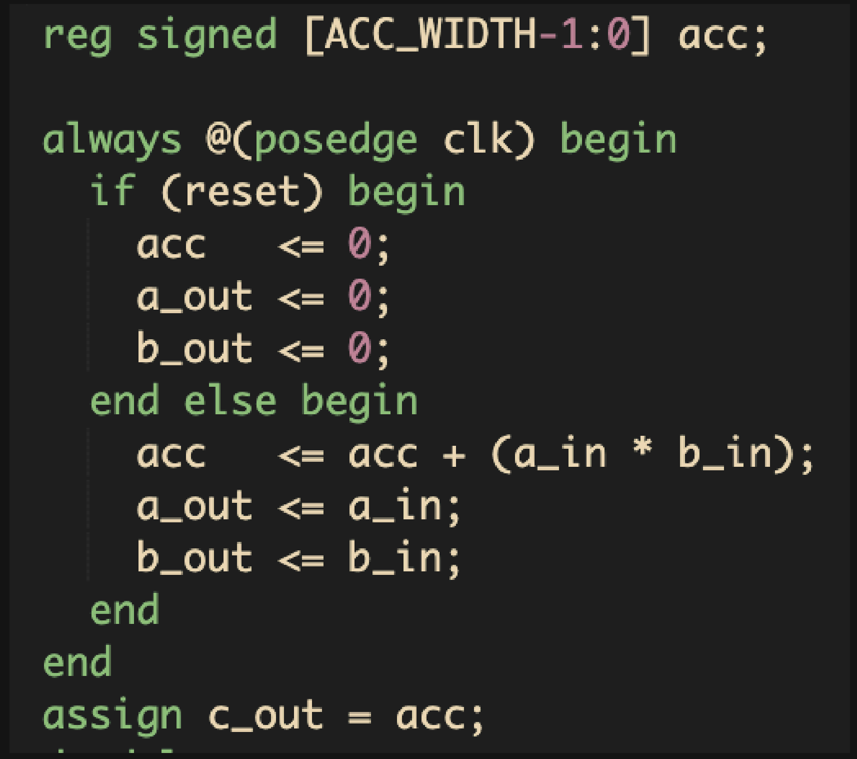
Key constraints that define the scope and design of the project :

- The system is designed to perform multiplication of two 8×8 matrices.
- Each matrix element is an 8-bit signed value.
- The multiply-accumulate (MAC) operations produce 16-bit signed results to accommodate the sum of products.
- The project assumes a single, constant operating mode with no dynamic scaling of array size or bit widths.
- It targets simulation and potential FPGA synthesis with fixed-size operands and predetermined matrix dimensions.

Processing Element (PE) – The Fundamental Unit







```
reg signed [ACC_WIDTH-1:0] acc;

always @(posedge clk) begin
  if (reset) begin
    acc   <= 0;
    a_out <= 0;
    b_out <= 0;
  end else begin
    acc   <= acc + (a_in * b_in);
    a_out <= a_in;
    b_out <= b_in;
  end
end
assign c_out = acc;
```

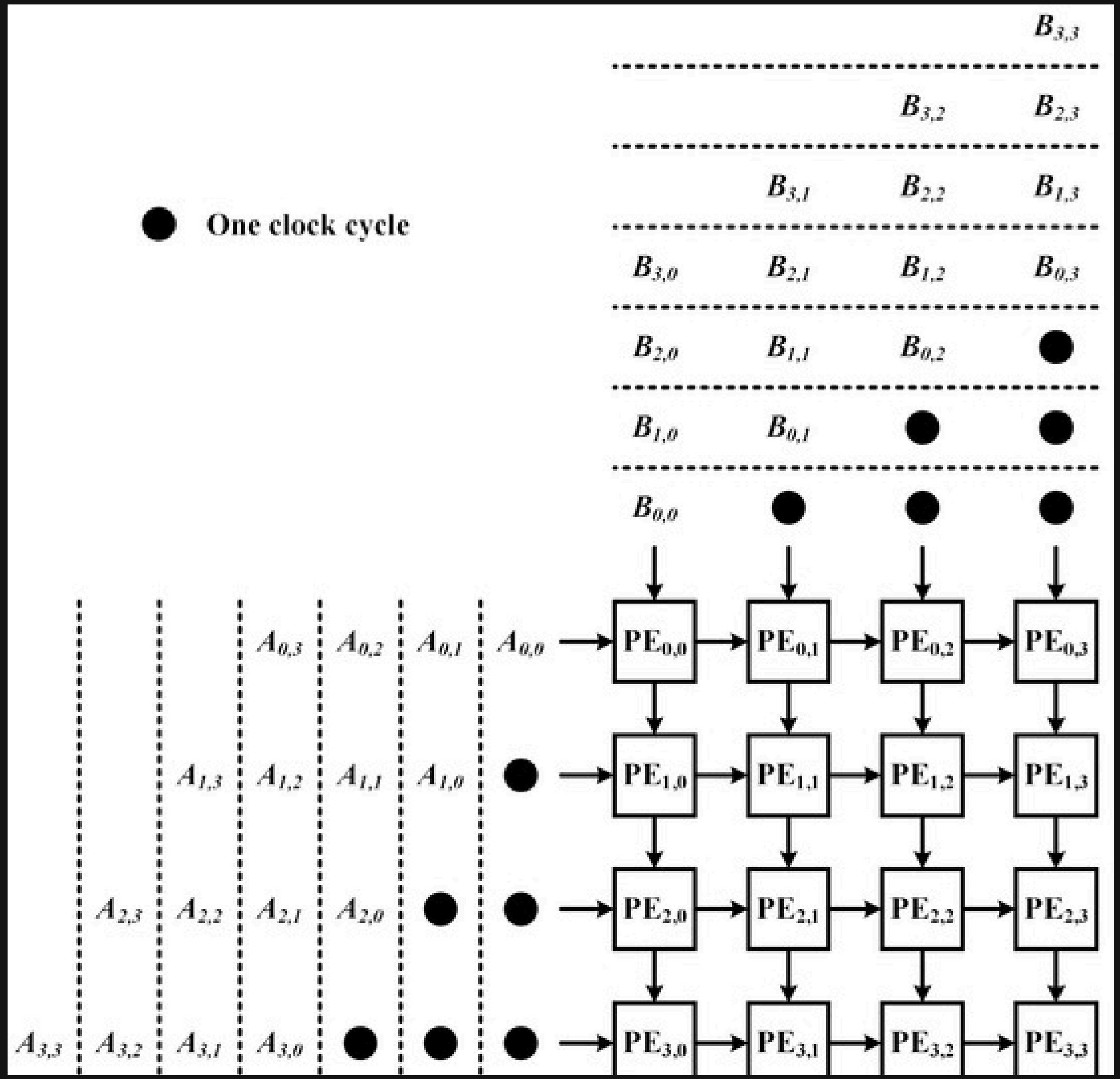
- The PE is the building block of the systolic array, responsible for performing the multiply-accumulate operation.
- The internal operation of the PE can be summarized in four steps that happen every clock cycle (after initialization) –

  Fetch → Multiply → Add → Forward

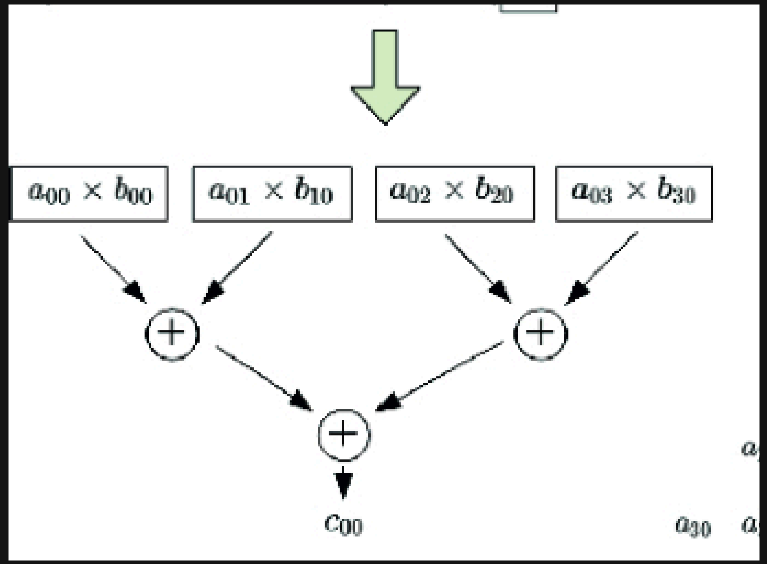
- In hardware terms, each PE contains a multiplier, an adder, and a few registers.
- By the design of the dataflow, each PE's accumulator will, by the end of the calculation, hold the sum of products for a unique element of the output matrix C.

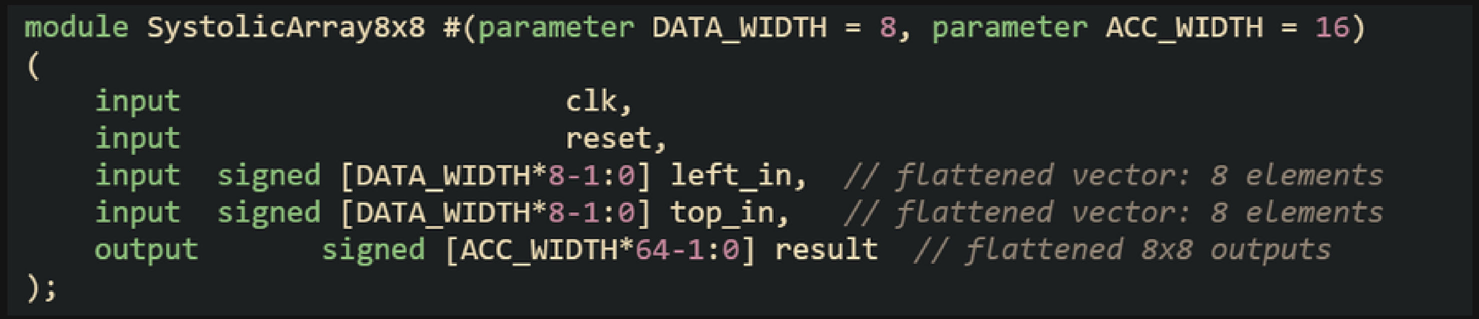
Systolic Array

$$\begin{pmatrix} a_{00} & a_{01} & a_{02} & a_{03} \\ a_{10} & a_{11} & a_{12} & a_{13} \\ a_{20} & a_{21} & a_{22} & a_{23} \\ a_{30} & a_{31} & a_{32} & a_{33} \end{pmatrix} \times \begin{pmatrix} b_{00} & b_{01} & b_{02} & b_{03} \\ b_{10} & b_{11} & b_{12} & b_{13} \\ b_{20} & b_{21} & b_{22} & b_{23} \\ b_{30} & b_{31} & b_{32} & b_{33} \end{pmatrix} = \begin{pmatrix} \boxed{c_{00}} & c_{01} & c_{02} & c_{03} \\ c_{10} & c_{11} & c_{12} & c_{13} \\ c_{20} & c_{21} & c_{22} & c_{23} \\ c_{30} & c_{31} & c_{32} & c_{33} \end{pmatrix}$$

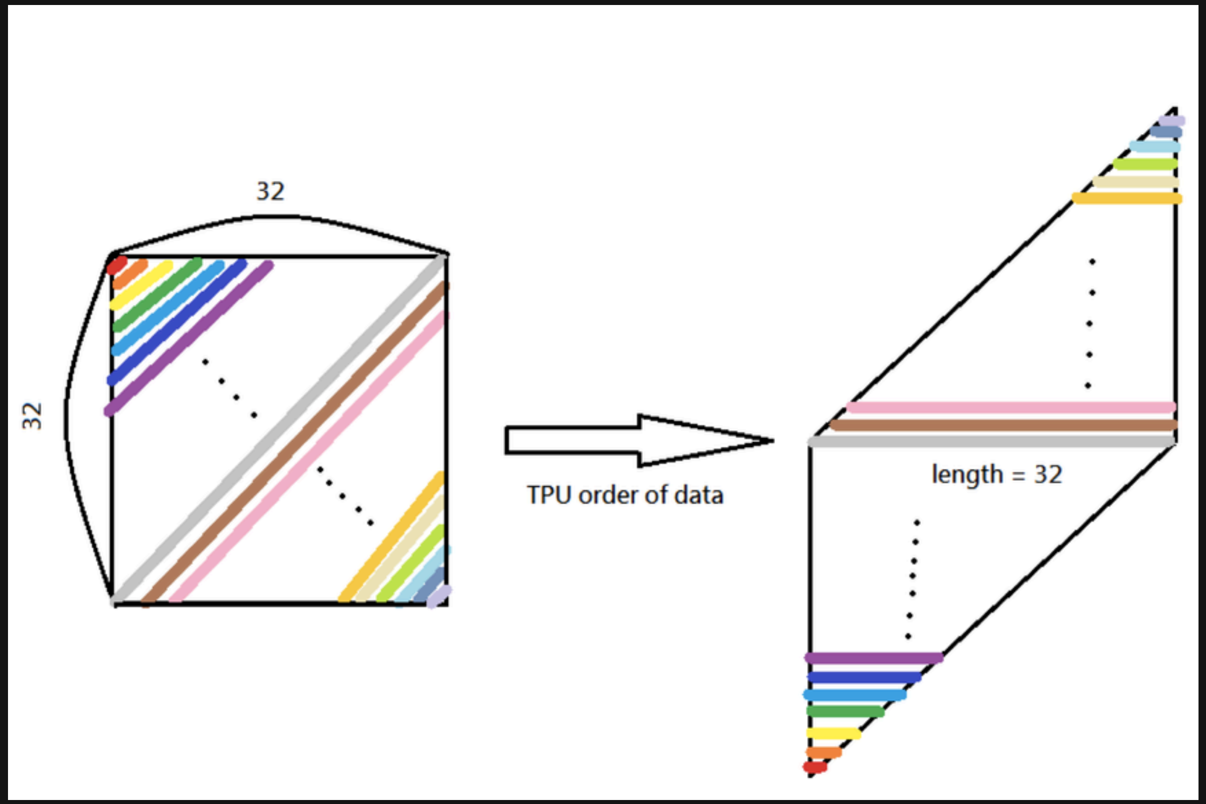


One clock cycle
$B_{3,3}$ $B_{3,2}$ $B_{2,3}$ $B_{3,1}$ $B_{2,2}$ $B_{1,3}$ $B_{3,0}$ $B_{2,1}$ $B_{1,2}$ $B_{0,3}$ $B_{2,0}$ $B_{1,1}$ $B_{0,2}$ $B_{1,0}$ $B_{0,1}$ $B_{0,0}$
$A_{0,3}$ $A_{0,2}$ $A_{0,1}$ $A_{0,0}$ $A_{1,3}$ $A_{1,2}$ $A_{1,1}$ $A_{1,0}$ $A_{2,3}$ $A_{2,2}$ $A_{2,1}$ $A_{2,0}$ $A_{3,3}$ $A_{3,2}$ $A_{3,1}$ $A_{3,0}$
$PE_{0,0}$ $PE_{0,1}$ $PE_{0,2}$ $PE_{0,3}$
$PE_{1,0}$ $PE_{1,1}$ $PE_{1,2}$ $PE_{1,3}$
$PE_{2,0}$ $PE_{2,1}$ $PE_{2,2}$ $PE_{2,3}$
$PE_{3,0}$ $PE_{3,1}$ $PE_{3,2}$ $PE_{3,3}$





```
module SystolicArray8x8 #(parameter DATA_WIDTH = 8, parameter ACC_WIDTH = 16)
(
    input                          clk,
    input                          reset,
    input  signed [DATA_WIDTH*8-1:0] left_in,  // flattened vector: 8 elements
    input  signed [DATA_WIDTH*8-1:0] top_in,   // flattened vector: 8 elements
    output       signed [ACC_WIDTH*64-1:0] result  // flattened 8x8 outputs
);
```



Conclusion

- We explored the architecture and importance of Tensor Processing Units (TPUs) in accelerating machine learning workloads.
- Compared CPUs, GPUs, and TPUs, highlighting their strengths and differences in handling complex computations.
- Demonstrated a custom matrix multiplication accelerator using modular Verilog components, simulating a simplified version of a TPU systolic array.
- Understood the design trade-offs and implementation details of Processing Elements (PEs) and systolic arrays.

Now we open the floor for some questions 😎

Thank You