Digital Design & Logic Synthesis
Course Project: Matrix Multiplication
1-Design

Mike Pines

January 2024

Contents

1 Introduction

Matrix multiplication on CPUs is complicated by the constraints imposed by their general purpose architectures and the limited number of internal registers. This can lead to memory bottlenecks during matrix multiplication, as the processor must frequently fetch data from memory, incurring latency that hampers overall performance. The inefficiencies in utilizing the limited internal registers can exacerbate the challenges associated with cache management, resulting in increased data movement between levels of the memory hierarchy.

Mitigating the challenges of matrix multiplication on CPUs often involves leveraging specialized hardware. Hardware accelerators, such as Graphics Processing Units (GPUs) or Tensor Processing Units (TPUs), are designed to excel at parallel processing, making them well-suited for intensive matrix operations. By offloading matrix multiplication tasks to these specialized units, the computational workload is distributed across a large number of cores, allowing for significant parallelism and faster execution.
These accelerators often come with a more extensive set of internal registers, reducing the need for frequent data exchanges with slower main memory. This integration of specialized hardware can dramatically enhance the overall performance of matrix multiplication, providing a scalable solution to address the computational demands of large-scale linear algebra operations , such as (to name one example) most machine-learning applications.

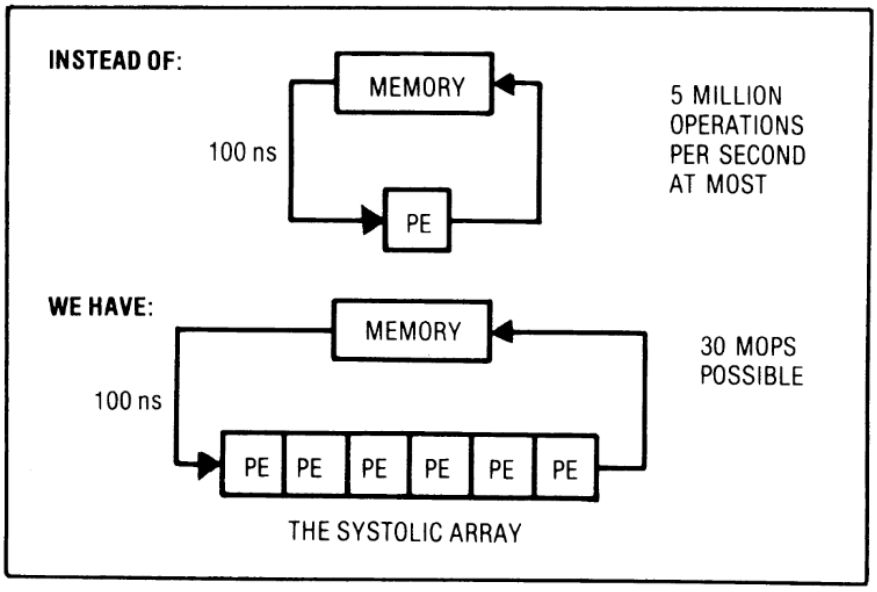

Figure 1: Basic principle of a systolic system [1]

In this project - your task is to design, verify and synthesize multiply or multiply-accumulate Processing Elements (PEs) arranged & controlled in a manner that would enable minimal data transfers.
It is strongly advised (**but not required**) to read the paper 'Why systolic architectures' by H.T. Kung referenced in this document.

2 Revision Notes

2.1 v1.1.1

- Added explanation for start-bit and mode-bit in the control register

2.2 v1.1

- Added this section
- Assigned bits [15:14] in the control register to operand reload (were 'reserved')
- Added a note in the 'Addresses' subsection to explain SP addressing
- Added a 'busy_o' output port to the top level interface
- Updated the SoC top-view picture to better reflect the required modules

3 Functional Description

The design is a hardware accelerator, controlled by sofrware running on the CPU.
Using this software, the CPU starts moving data from memory into the design using the AHB interconnect which is in turn translated into
APB transactions.
An APB-slave (that you implement) inside the design further
translates the transactions into register read/writes.
To implement this APB-slave you must read the
AMBA APB4 protocol specification [2]
(PPROT can be ignored).
Once the design has been activated, it calculates the results in
$[T : n \cdot k \cdot m]$ cycles (what is the lower bound T?) and stores in to SP.

4 Operation

The goal of this project is to efficiently compute un/biased matrix multiplication:

$$D = A \times B + C$$

$$\begin{bmatrix} a_{1,1} & a_{1,2} & ... & a_{1,k} \\ a_{2,1} & a_{2,2} & ... & a_{2,k} \\ \vdots & \vdots & \ddots & \vdots \\ a_{n,1} & a_{n,2} & ... & a_{n,k} \end{bmatrix} \times \begin{bmatrix} b_{1,1} & b_{1,2} & ... & b_{1,m} \\ b_{2,1} & b_{2,2} & ... & b_{2,m} \\ \vdots & \vdots & \ddots & \vdots \\ b_{k,1} & b_{k,2} & ... & b_{k,m} \end{bmatrix} + \begin{bmatrix} c_{1,1} & c_{1,2} & ... & c_{1,m} \\ c_{2,1} & c_{2,2} & ... & c_{2,m} \\ \vdots & \vdots & \ddots & \vdots \\ c_{n,1} & c_{n,2} & ... & c_{n,m} \end{bmatrix} =$$

$$D = \begin{bmatrix} \sum_{i=1}^{k} a_{1,i}b_{i,1} + c_{1,1} & \sum_{i=1}^{k} a_{1,i}b_{i,2} + c_{1,2} & ... & \sum_{i=1}^{k} a_{1,i}b_{i,m} + c_{1,m} \\ \sum_{i=1}^{k} a_{2,i}b_{i,1} + c_{2,1} & \sum_{i=1}^{k} a_{2,i}b_{i,2} + c_{2,2} & ... & \sum_{i=1}^{k} a_{2,i}b_{i,m} + c_{2,m} \\ \vdots & \vdots & \ddots & \vdots \\ \sum_{i=1}^{k} a_{n,i}b_{i,1} + c_{n,1} & \sum_{i=1}^{k} a_{n,i}b_{i,2} + c_{n,2} & ... & \sum_{i=1}^{k} a_{n,i}b_{i,m} + c_{n,m} \end{bmatrix}$$

Data for matrices A and B are written to operand registers using the APB.

Data for matrix C are optionally read from an internal data store called scratchpad memory, which holds previous results.

You will receive control & data from the APB bus, feeding your designs operand registers, the result needs to be saved inside the designs ScratchPad (SP) memory.

4.1 Example configuration

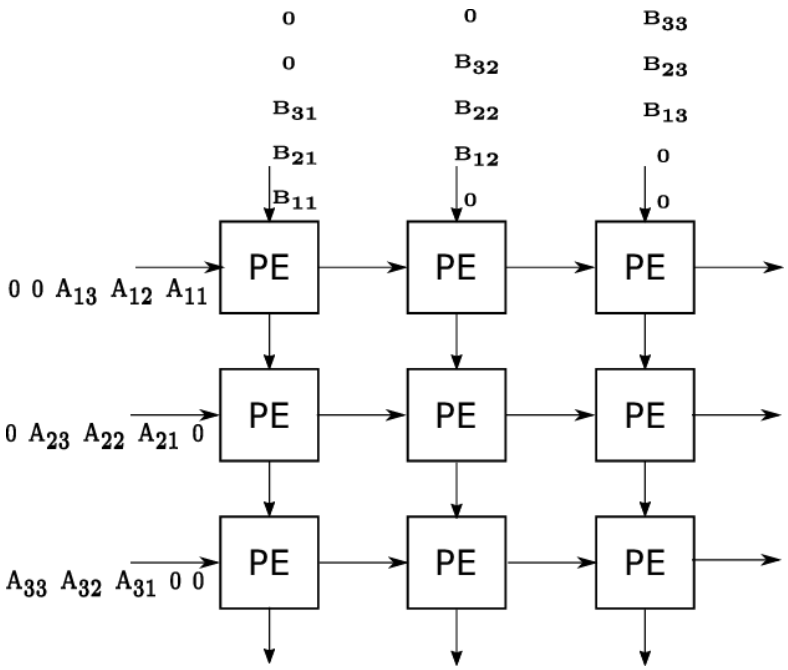

Figure 2: Processing Elements in a systolic matrix topology

Figure-2 shows a proposed approach for a mid-level design.
On the first compute cycle, only the element from the first row / first column enter $PE_{1,1}$.
On the 2nd compute cycle, elements are fed into $PE_{1,1}$, $PE_{2,1}$ and $PE_{1,2}$ etc'...

4.2 Requirements

Minimal data movement between 'main memory' and the accelerator is achieved by requiring that each data element is moved in to the design only once and used in all relevant operations.

Your design should:

- Have reconfigurable n,k and m (≤ 4).
- Hold and/or use intermediate results when required.
- Detect overflows (Signed operation).
- Report errors using pslverr_o

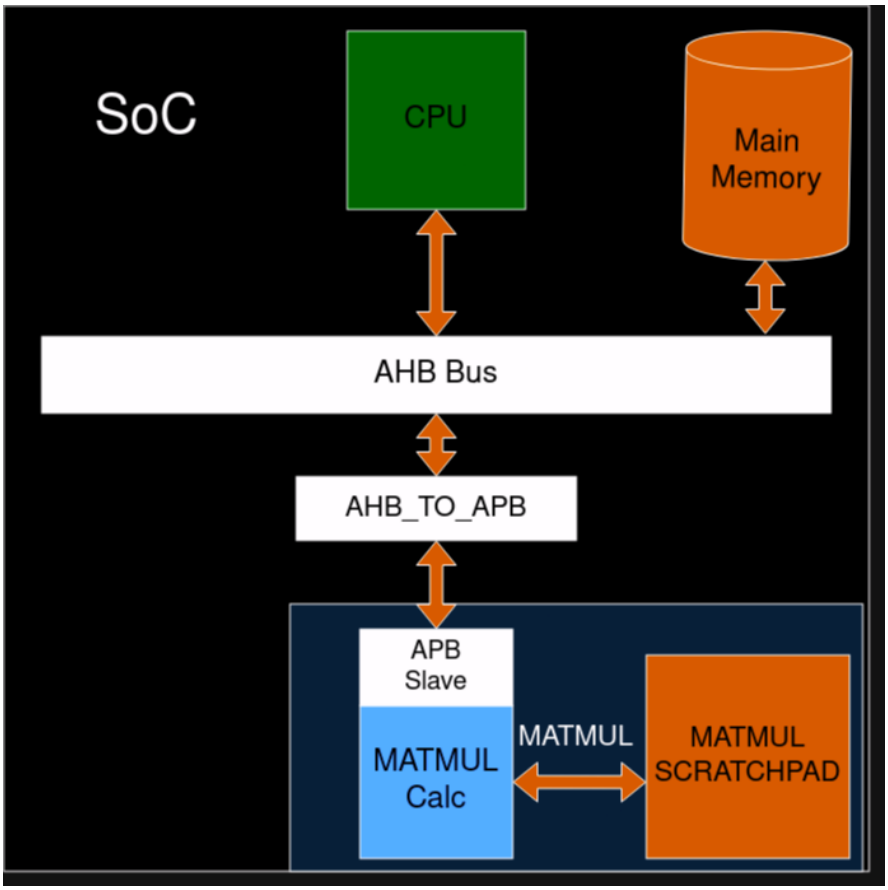

Figure 3: SoC Overview

5 Tables

5.1 Parameters

The following parameters and values should be supported:

Parameters			
Parameter Name	Values	Description	Constraints
DATA_WIDTH (DW)	8, 16, 32	Bit-Width of a single element	$\leq \frac{BW}{2}$
BUS_WIDTH (BW)	16, 32, 64	APB Bus data bit-width	$BW\%DW = 0$
ADDR_WIDTH (AW)	16, 24, 32	APB address space bit-width	-
SP_NTARGETS (SPN)	1, 2, 4	The number of addressable targets in SP	-

To simplify, the maximal dimension (MAX_DIM) of your matrix multiplication hardware should be

$$MAX\_DIM = \frac{BUS\_WIDTH}{DATA\_WIDTH}$$

hence the requirement:

$$BUS\_WIDTH\%DATA\_WIDTH = 0$$

Clarification: the data-type that your design will operate on is DATA_WIDTH bits wide.

5.2 Top-Level Interface

Interface			
Signal(s)	Mode	Bounds	Notes
clk_i	input	-	Clock signal for the design
rst_ni	input	-	Reset signal, active low
psel_i	input	-	APB select
penable_i	input	-	APB enable
pwrite_i	input	-	APB write enable
pstrb_i	input	[MAX_DIM-1:0]	APB write strobe ('byte' select)
pwdata_i	input	[BUS_WIDTH-1:0]	APB write data
paddr_i	input	[ADDR_WIDTH-1:0]	APB address
pready_o	output	-	APB slave ready
pslverr_o	output	-	APB slave error
prdata_o	output	[BUS_WIDTH-1:0]	APB read data
busy_o	output	-	Busy signal, indicating the design cannot be written to.

5.3 Addresses

The following addresses should be assigned:

Registers			
Register Name	Width	Address	Notes
Control	16	0	RW, Sub-mapping described in the next section
Operand-A	$DATA\_WIDTH \cdot MAX\_DIM^2$	4	RW, Sub-addressing(*)
Operand-B	$DATA\_WIDTH \cdot MAX\_DIM^2$	8	RW, Sub-addressing(*)
Flags	MAX_DIM^2	12	RO (from the APB).
Scratchpad	$SPN \cdot MAX\_DIM^2 \cdot BUS\_WIDTH$	16+:(SPN*4)	RO (from the APB)(**).

(*) : Sub-addressing for operand matrices depends on the dataflow type you choose to implement (if any).
Static elements need to be element-addressable, assuming from the constraint on BW that a single write fills a single row/column, than $log_2(MAX\_DIM)$ extra address bits are required (paddr_i[5+:$log_2(MAX\_DIM)$]).
Dynamic elements can be fed from a FIFO that needs no extra address bits.
(**) : Sub-addressing from paddr_i[5+:$2 * log_2(MAX\_DIM)$].

6 Registers

6.1 Control Register (Address 0)

Holds the writeable control signals for the design such as the start-bit, mode of operation and operand dimensions.
A write to this register during operation raises an error (pslverr_o is asserted).

- Start Bit (Bit 0) - Starts operation, the design is inactive when low, stays asserted for the duration of operation and is de-asserted by the design upon completion, writes to the design when this bit is set return an error.
- Mode bit (Bit 1) - Enables biased operation when asserted.
- Write Target (Bits 3:2), the destination inside SP
- Read Target (Bits 5:4), the source in SP. (Only read if Mode-Bit is set)
- DataFlow type (Bits 7:6), **optional** & read-only if one of the following is implemented:
  - None implemented : 2'd0
  - Output-Static : 2'd1
  - Weight-Static : 2'd2
  - Other, well-defined dataflow type : 2'd3
- Dimension-N (Bits 9:8), the 1st dimension of matrix A and the 1st dimension of out matrix C.
- Dimension-K (Bits 11:10), the 2nd dimension of matrix A and the 1st dimension of matrix B.
- Dimension-M (Bits 13:12), the 2nd dimension of matrix B and the 2nd dimension of out matrix C.
- Reload Operand-A (Bit 14), specifies the 1st operand should be reloaded upon completion. (*)
- Reload Operand-B (Bit 15), specifies the 2nd operand should be reloaded upon completion. (*)

(*) : Only required if using buffers/queues.

6.2 Operand Registers

6.2.1 Matrix-A (A) Registers (Address 4)

These hold values for Matrix-A that are going to be/are being fed into the design.
Read addresses for these registers are generated during operation, are readable at all times and writeable when inactive.

6.2.2 Matrix-B (B) Registers (Address 8)

These hold values for Matrix-B that are going to be/are being fed into the design.
Read addresses for these registers are generated during operation, are readable at all times and writeable when inactive.

6.3 Result Flags (Address 12)

Registers that hold flags indicating if an over/under flow occured during calculation of each output cell.

6.4 ScratchPad (SP) Memory (Address 16)

Fast memory to store results, the SP should be readable from the APB and only be writeable from the design, it should allow reading bias values when the mode bit is set to biased operation.

7 Submission Requirements

- HDL-Designer library including the **project file**, **hdl** & **hds** folders.
- The top level module should be called **matmul** and the file **matmul.v**, written in Verilog-2005.
- Short design document (5-10 pages) with the following:
  - Block diagram of the design.
  - Functional description of the design.
  - Flow-chart / State-machine diagram.
  - Explanation for all rules removed from the design-checker ruleset.

8 Submission

- Attach the document to the top module in HDL-Designer using the 'Side Data' panel.
- Delete the 'work' library.
- Compress the project file, hds and hdl directories into a zip file named <ID1>_<ID2>.zip where ID1 & ID2 are the teudat-zehut numbers of the pair.
- Submitted to Moodle by **both** students.

Submission Date : 01/02/24 at 23:59

Late submissions incur a penalty of 5 points per day.

9 Submission Evaluation

This part (design) is 30% of the **total projects grade**.

9.1 Preliminary Test

Your code must **compile** without errors in HDL-Designer and QuestaSim.
Failure in compilation means a 0 grade on this part of the project.

9.2 Checks & Grading

Grade-%	Task	Description
40%	RTL code quality	Graded from Design-Checker using the policy (DO-254 +RMM) used in this course. Any excluded rules must have a good reason and must be documented.
30%	Documentation	Block diagrams, flow diagram, FSM diagrams easy to read and understand your design (use Digital Block_02_Definition_Template). RTL should have enough comments to explain every part of the design.
30%	Reuse	Design compiles using every parameter value described in the parameters table.

References

[1] H.T. Kung (1982) *Why Systolic Architectures?*

[2] *ARM - AMBA APB4 specification*