Matrix Multiplication

1-Design

matmul

Digital High-Level Design

Version 0.1

Classification:	**Title:**	Owner	Creation Date	Page
Logic Design course	Course project: Matrix Multiplication	Nadav Rozenfeld Noam Guez	27, January, 2024	1 of 19

Revision Log

Rev	Change	Description	Reason for change	Done By	Date
0.1	Initial document				
0.2					
0.3					

Table of Content

LIST OF FIGURES

LIST OF TABLES

ABSTRACT OF THE PROJECT

This project develops an enhanced matrix multiplication and addition accelerator, designed using Verilog. Functioning as an APB slave (refer to Reference 1), this module facilitates both writing to and reading from the accelerator. This includes writing matrices or control register data, as well as retrieving multiplication results and an overflow flag matrix. The core computation employs the systolic multiplication method (refer to Reference 2), adhering to the specifications outlined in the mission brief (refer to Reference 3). This design aims to optimize computational efficiency and accuracy in matrix operations.

1. BLOCKS FUNCTIONAL DESCRIPTIONS

In this section, we detail the various components of our project, outlining their functions and roles. Adopting a bottom-up approach, our explanation begins with the Processing Element (PE), the fundamental unit of computation. Next, we delve into the APB slave communication unit, encompassing both the writing mechanism and the primary slave system. Following this, we describe the FIFO module, which serves as the intermediary between the APB and the PE. Lastly, we discuss the overarching MATMUL module, integrating all these elements into a cohesive whole. This structured exposition aims to provide a clear understanding of each component and its contribution to the project's functionality.

1.1.1 PE

The PE is the fundamental unit for the systolic architecture. Each PE has an accumulator which is initialized with zeros when reset is low. Each clock cycle the value – **ain** * **bin** is added to the sum only if the **start** bit is high. The **result** value is determined by the following asynchronous logic-

If **mode** is high: **result** = sum + **cin**

Otherwise: **result** = sum

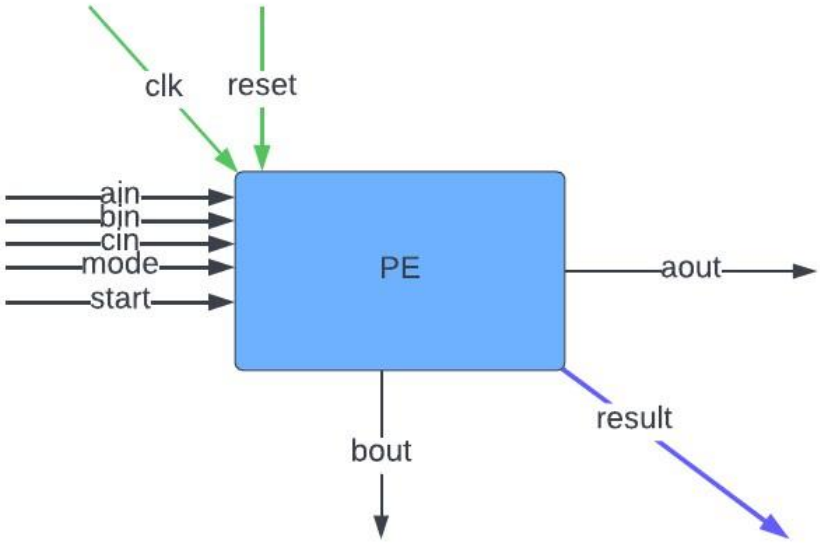

Figure 1: PE block diagram

Name	Mode	Type	Signed	Bounds
ain	input	wire		[DATA_WIDTH-1:0]
bin	input	wire		[DATA_WIDTH-1:0]
reset	input	wire		
clk	input	wire		
start	input	wire		
mode	input	wire		
cin	input	wire		[DATA_WIDTH-1:0]
aout	output	reg		[DATA_WIDTH-1:0]
bout	output	reg		[DATA_WIDTH-1:0]
result_out	output	reg		[2*DATA_WIDTH:0]

Table 1: PE interface.

1.1.2 APB WRITER

Data:

The APB WRITER is a component inside the APB slave that is responsible for the writing operations according to AMBA APB protocol. The input **pwdata** is sampled to a register in every cycle that both **psel** and **penable** are not high (before the AMBA APB access phase of the transfer). In cycles that **psel** and **penable** are high, the most LSB DATA_WIDTH bits of this register are shifted out to the output **wdata_out**. The output **wdata_out** is typically an output towards the inner parts of the design that are writable.

Addressing:

Since MAX_DIM can be at most 4, there are 8 FIFO components that feed the systolic archtitecture as described in figure 3. Together with the control there are 9 writable addresses in the top MATMUL design. The output signal **locals_enable_vec** is a chip select vector that is determined according to **paddr** input.

locals_enable_vec is a 9 sized vector that in every given moment only one bit of it is set and the others are low (one-hot). Each bit is used as a local enable to one of the 9 writeble targets.

Data Enabling:

According to AMBA APB protocol, a given **wdata_out** can be either a matrix data element or a "don't - care" value, the input vector **pstrb** is used as global data enable. I.E a writble target is actually being write if both the global and local enables are set. **pstrb** is sampled and registered before the AMBA APB access phase of the transfer . In every cycle during the access phase the LSB of the strb inner register is shifted out to be the global enable signal **global_en_wr.**

End of transaction:

According to AMBA APB protocol, pready signal should be set if there is only 1 cycle left to the trasmition. Therefore pready is implemented to be set whenever the strb inner register contains the value of 1 (I.E last data element in this transaction is being shifting out to **wdata_out**).

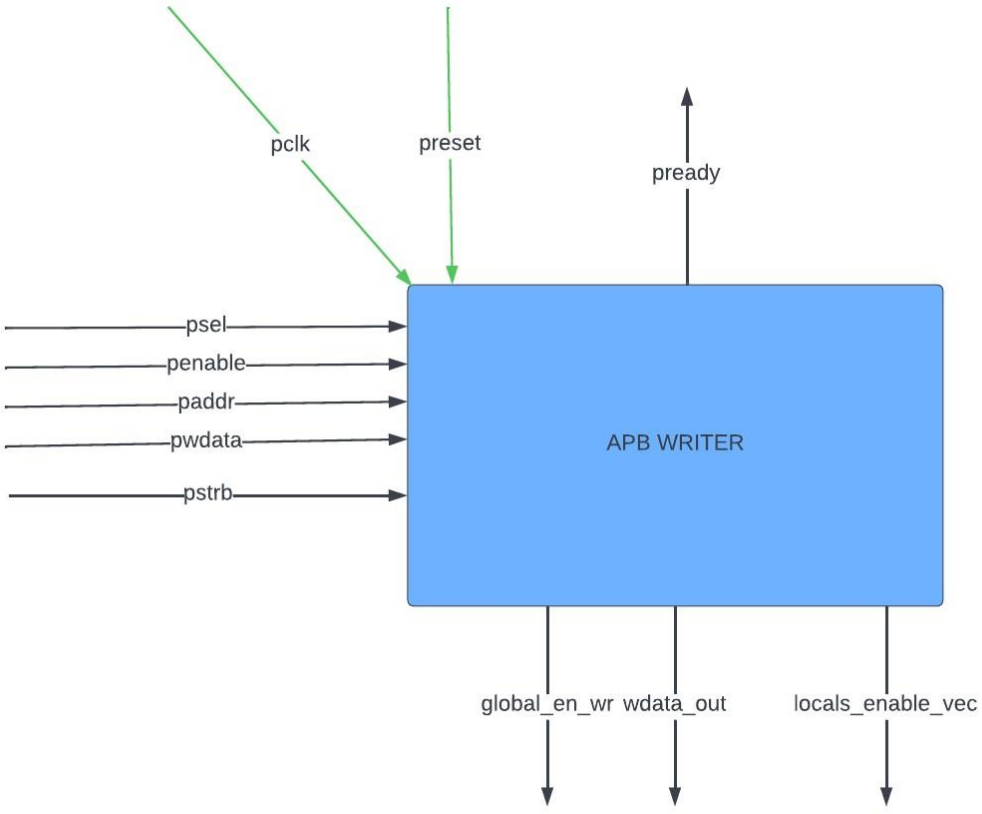

Figure 2: APB SLAVE WRITER block diagram

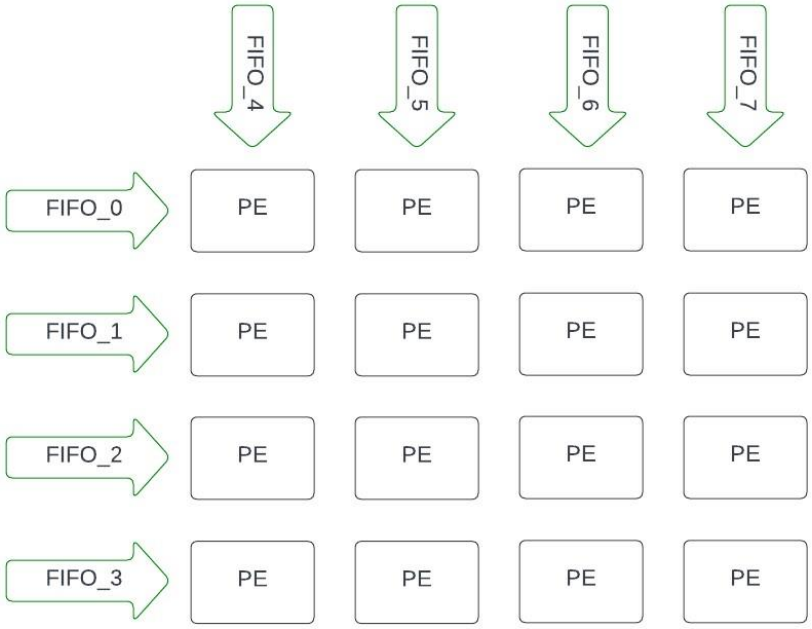

Figure 3: MATMUL systolic architecture

B	C	D	E	F
Name	Mode	Type	Signed	Bounds
pclk	input	wire		
preset	input	wire		
paddr	input	wire		[ADDR_WIDTH-1:0]
psel	input	wire		
penable	input	wire		
pwdata	input	wire		[BUS_WIDTH -1:0]
pstrb	input	wire		[MAX_DIM -1:0]
pready	output	reg		
locals_enable_vec	output	reg		[NUM_OF_FIFOS:0]
wdata_out	output	reg		[DATA_WIDTH-1:0]
global_en_wr	output	reg		

Table 2: Apb_slave_writer interface.

1.1.3 APB SLAVE

The APB slave is compromised from both writing logic and reading logic. The writing logic is implemented by the apb writer component (described earlier). The reading component will be implemented in the apb slave module itself (see important note in appendix). The output pready signal will be logical OR between the generated pready from the writer component and the pready that is produced from the reading logic. The global_en_wr output of the top slave will be logical AND between the generated global_en_wr from the writer and pwrite input signal.

In our implementation, the A and B matrix memories were established within this module, primarily to facilitate reading these matrices. It's important to note that these memories are not critical for the actual computation process. This is because we have incorporated FIFO to dynamically feed data into the calculation, a feature we will elaborate on shortly. This design choice underscores our focus on efficient data handling and streamlined computational flow.

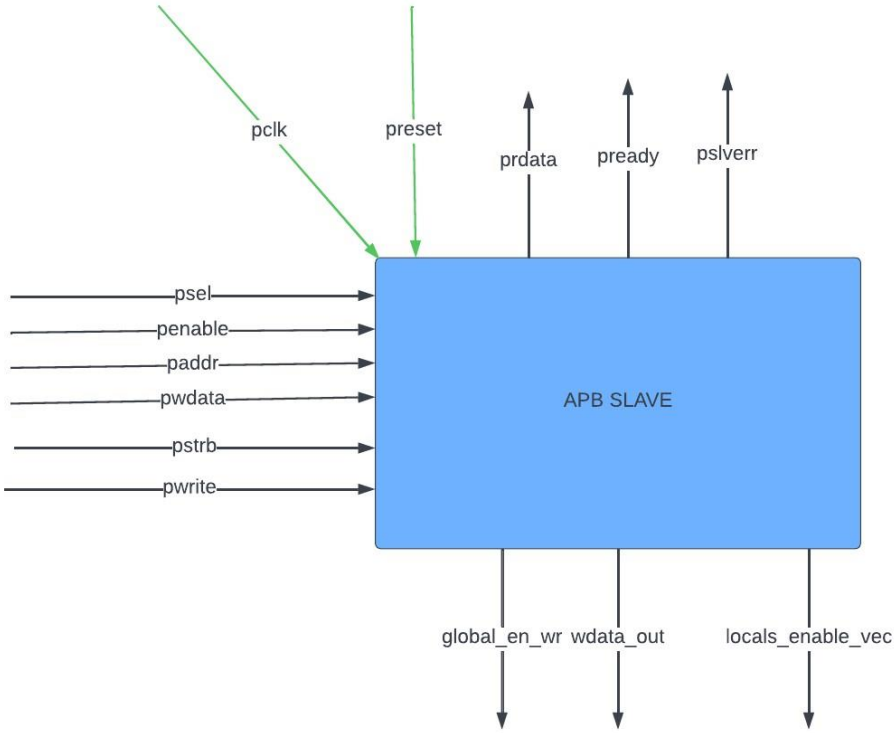

Figure 4: APB_SLAVE block diagram

B	C	D	E	F
Name	Mode	Type	Signed	Bounds
pclk	input	wire		
preset	input	wire		
paddr	input	wire		[ADDR_WIDTH-1:0]
psel	input	wire		
penable	input	wire		
pwrite	input	wire		
pwdata	input	wire		[BUS_WIDTH -1:0]
pstrb	input	wire		[MAX_DIM -1:0]
pready	output	wire		
pslverr	output	wire		
prdata	output	wire		[BUS_WIDTH -1:0]
write_locals_enable_vec_out	output	wire		[NUM_OF_FIFOS:0]
wdata_out	output	wire		[DATA_WIDTH-1:0]
global_en_wr_out	output	wire		

Table 3: apb_slave interface.

1.1.4 FIFO

We used 8 “fifo” elements for feeding the unique PE calculation units of the systolic architecture. When **enable_write** is high, the **data_in** element is wrriten to an internal queue register of the component by the receiving order. The input “**placement_in**” is used for placing zeroes at the beginning of the queue for each unique PE (such as described in figure 6). For example, a fifo element for the second row/column as described in the green square will have **placement_in** = 1, because there is a need for single zero at the beginning of the queue. When **start** is driven high (from the control register of the top design), the fifo starts feeding it’s designated PE element by putting the LSB DATA_WIDTH of the queue on data_out and shifting left. Eventually the fifo will contain only zeroes and **data_out** won’t influence the PE calculation anymore.

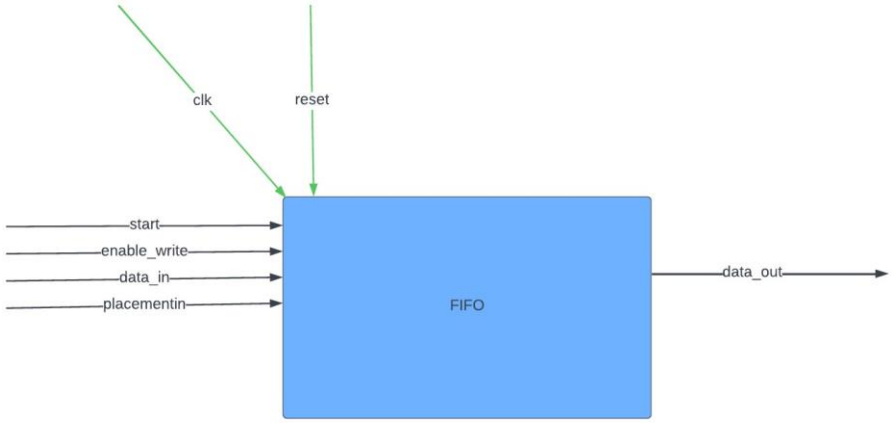

Figure 5: FIFO block diagram

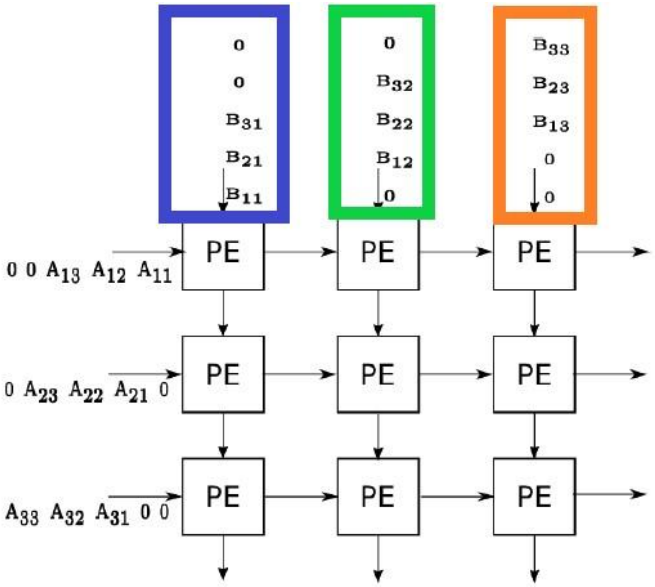

Figure 6: Systolic architecture – PE's "feeders" implemented by fifo components

Name	Mode	Type	Signed	Bounds
clk	input	wire		
reset	input	wire		
start	input	wire		
enable_write	input	wire		
data_in	input	wire		[DATA_WIDTH-1:0]
placementin	input	wire		[MAX_DIM/2:0]
data_out	output	reg		[DATA_WIDTH-1:0]

Table 4: fifo interface.

1.1.5 MATMUL

This top module is the main component of our design, integrating 16 PE and 8 FIFOs, as illustrated in Figure 3. It also encompasses an APB slave, inclusive of the APB writer, the A and B matrices. Additionally, this module incorporates several key elements: a control register, a scratchpad, and a result flags matrix. We will first briefly describe the functions of these three components before delving into the logic of the matrix multiplication (matmul).

a. Control Register: This register is pivotal for managing the calculation process. It includes a start bit [0] to initiate the calculation, a mode bit [1] to determine whether the calculation includes a bias addition, and fields [3:2] and [5:4] for specifying the write and read targets, which dictate where in the Scratchpad the results should be stored and from where the bias matrix C should be read. It also defines the Data Flow type [7:6] (in our case, set to '2'd1' indicating an output static data flow), and the dimensions N, K, and M of the matrices [9:8], [11:10], [13:12]. The two MSBs are undefined and unused. Users can read from and write to this register through the APB slave at address 0.

b. Scratchpad: This component is the module's fast, private memory, capable of storing 1, 2, or 4 matrices based on the SP_NTARGETS parameter. It holds the result matrix D and, if specified by the mode bit, retrieves the bias matrix C. This memory is inaccessible for writing by the user but can be read to extract solutions via the APB slave at address 16.

c. Results Flags: This matrix monitors the validity of the calculations, particularly watching for any overflow or underflow incidents. Each PE is associated with a specific flag in this matrix. Like the Scratchpad, this matrix is not writable by the user but can be read for solution analysis through the APB slave at address 12.

Having outlined these components, let's proceed to examine the Matmul block diagram and the I/O table to understand its operational functionality.

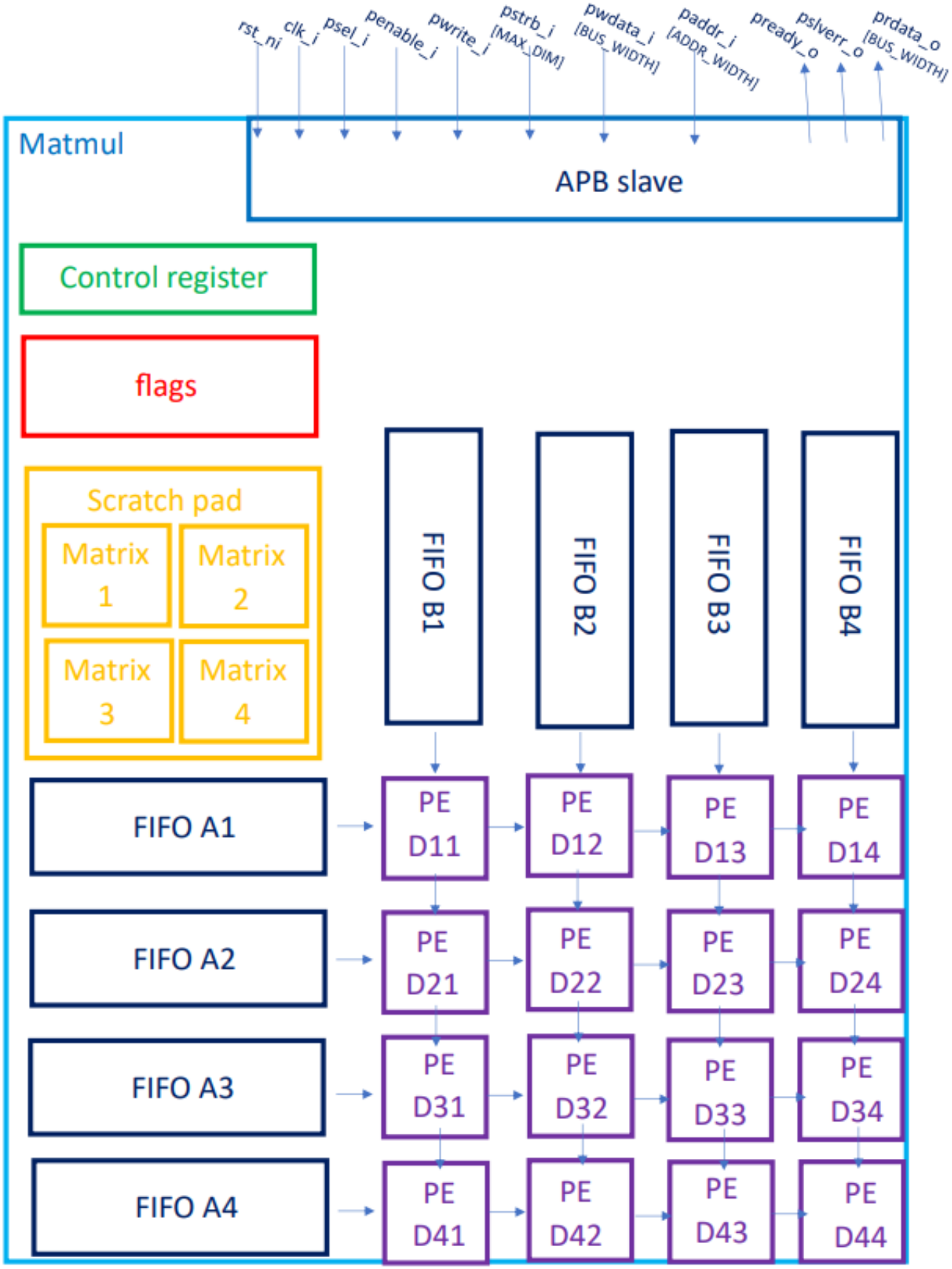

Figure 7: Matmul main blocking diagram

B	C	D	E	F
Name	Mode	Type	Signed	Bounds
pclk	input	wire		
preset	input	wire		
paddr	input	wire		[ADDR_WIDTH-1:0]
psel	input	wire		
penable	input	wire		
pwrite	input	wire		
pwdata	input	wire		[BUS_WIDTH -1:0]
pstrb	input	wire		[MAX_DIM -1:0]
pready	output	wire		
pslverr	output	wire		
prdata	output	wire		[BUS_WIDTH -1:0]

Table 5: Matmul inputs and outputs

Let's now explore the accelerator process from a top-down perspective:

1. APB Slave Input: Initially, the APB slave receives information about the A and B matrices. It is responsible for assigning this data to the FIFOs.

2. Data Organization via FIFOs: Once the matrices' data is in the FIFOs, these units organize it to be optimally prepared for the systolic calculation process.

3. Starting the Calculation: The calculation is triggered when the module detects the start bit set in the written control register. This is the signal for the computational process to commence.

4. Incorporating Bias (If Applicable): If the mode bit is set to 1, each Processing Element (PE) retrieves its respective C bias from the Scratchpad at the specified read target. This bias is then integrated into the ongoing calculation.

5. Systolic and Calculation Duration: This accelerator utilizes a systolic architecture for fixed output matrix calculations, streamlining the data flow and computational efficiency. Inputs are assigned once to the FIFOs, after which the PEs synchronously pass the data for precise computation of the matrix D. This approach ensures minimal data redundancy and maximizes processing speed, making it highly effective for matrix operations. For an in-depth understanding, please see the referenced material on systolic architecture. As defined in the specifications (refer to references), the entire process takes T = NKM clock cycles. This duration is a function of the dimensions and complexity of the matrices involved.

6. Writing Results and Flagging Overflows: The outcomes of the computations are written to the Scratchpad at the designated write target. Simultaneously, the most significant bit (MSB) of each result is sent to the Result Flags. This is to indicate whether any overflow has occurred during the calculations.

7. User Interaction and Continuation: Post-calculation, users can access the solutions by reading from the Scratchpad. They also have the option to input new A and B matrices and initiate a new round of calculations. In such cases, the previously obtained results can be utilized as a bias for these subsequent computations.

This structured process ensures efficient data handling, precise calculations, and user-friendly interaction with the accelerator, allowing for a smooth and dynamic computational experience.

2. APPENDIX

2.1 Disabled Rules

RULE	**PLACE**	**JUSTIFICATION**
All commands should not be in separate lines.	The whole design	No hardware meaning.
Asynchronous reset is not synchronized before being used.	The whole design	This is quite a simple and short design. No necessary to create a sync logic for the async resets because the influence is relatively small.
Net 'paddr[31:5]' is unused.	The whole design	We use only 5 LSB bits of the address.
Do not use disallowed character: HT (decimal 9)	The whole design	No hardware meaning. We use tab for documentation.
Bit widths differ on left (32)	matmul.v lines 81, 82,	Used in integers in for loops.

and right (33) of assignment.	,100-102,108,109,191,192 apb_slave.v lines 90,91	There is no need for the overflow bit
Nesting at an assignment statement exceeds the maximum of 3	matmul.v lines 90, 103, 110	Writing to a 3- dimensional array (an array of matrixes for scarchpad implementation), there is a need for a nested for loop
Genvar "q1" should be assigned before being read	matmul.v lines 149, 162	Genvars are actually assigned before reading (in the for loop definition).
Avoid using hard coded numeric values such as "[3:2]" for specifying ranges of the multi-bit operand "control_reg"	matmul.v lines 53-57, 73, 149, 162 apb_slave.v line 85	matmul.v 53-57, 73: Specified bit in the control register (the control register defenition is not generic). matmul.v 149, 162: we used only 2 LSB of the indexes of the for loop. apb_slave.v 85 we used 2 bits counter vector only
'control_reg[7:6]' is never used (read from).	matmul.v line 41	Bits 7:6 of the control register are not used by spec defenition
'errsig' is never used (read from) or assigned (written to). // Net 'flags[3:0]' is unused.// 'dimension_n' is never used (read from). And more such messages (lines are elaborated)	matmul.v lines 42-43,55,56,67-68 apb_slave.v lines 33, 34	We haven't implemented yet the reading logic of the design, and the errors logic (due to time limitations).

2.2 An Important Note

The following utilities defined by the spec are not currently implemented due to time limitations:

1. Reading ability – The apb slave's readding ability is not implemented yet. Therefore, some signals (like prdata) are temporarily not connected/used.

2. Error handling – The ability of error indicating is not implemented yet. Therefore the signal pslverr is temporary high Z.

2.3 Terminology

LSB - Least Significant Bit

TBR - To Be Reviewed

TBD - To Be Defined

2.4 References

1. ARM - AMBA APB4 specification

2. H.T. Kung (1982) Why Systolic Architectures?

3. SPEC of the project 2023/4