Project 2: Systolic Array Design

Systolic Array GitHub Repository: https://github.com/karans5/systolic_array
MAC GitHub Repository: https://github.com/karans5/mac

Malavika Sanjeevan

EE24M098

Karan S

NS24Z028

Systolic Array Design

A systolic array is a parallel processing architecture commonly used for matrix multiplications.
In this implementation, the systolic array consists of 16 MAC units arranged in a 4x4 grid. The design uses pipelined data flow, where the input data for matrix A flows horizontally across the array, while the input data for matrix B flows vertically. The intermediate results propagate between MAC units, enabling an efficient computation. The module uses FIFO queues to buffer input data for matrices A, B and an optional bias matrix C. A single rule systolic step helps govern the data flow, ensuring the data is processed synchronously across all MAC units. At the end of the computation, the results are retrieved from the last row of the systolic array.

Interface:

Contains the method for interacting with the systolic array:

a. INPUT METHODS:
   i. **put_A_rowi** : Pushes values for row ‘i’ of matrix A. There are four sets of push methods: put_A_row0, put_A_row1, put_A_row2, and put_A_row3. The A values flow horizontally across the systolic array, starting from the respective rows. Each method enqueues a single A value into a FIFO buffet for the respective row. This buffer ensures data is passed properly to the next MAC units in the correct sequence.
   ii. **put_B_colj** : Pushes values for column ‘j’ of matrix B. Similar to the put_A_rowi method, each column has four sets of put_B_col methods. The B values flow vertically down each column of the systolic array. Each method enqueues a single B value into a FIFO buffer for the respective column. The buffered B values are processed by the MACs column by column.
   iii. **put_select_colj**: Push select values to control the computation. Each column has its own ‘select’ input to allow control over computation in that column. The select value can take either a ‘1’ or a ‘0’.
   iv. **put_C_colj** : Push initial values for column ‘j’ of matrix C for accumulation. The C values are used to initialize the accumulation process within the MAC. For the first rows of MACs, C values are directly fetched from the FIFO buffers using these methods. For subsequent rows, C values are propagated from the results of the row above.

b. OUTPUT METHODS:
   i. get_MAC_results: Retrieve results from the last row of the systolic array. The final MAC results are stored in a vector of 4 values, each corresponding to one column in the last row.

Module:

a. **`Vector#(4,Vector#(4,MAC_top_ifc))mac_array<-replicateM(replicateM(mk_MAC_top));`**

A **2D vector** (array) is used to model the 4x4 grid.

- Each row of the grid is a vector of 4 MAC units.
- `MAC_top_ifc`: This interface defines the operations each MAC unit can perform (e.g., receive inputs, compute, and pass results).

`replicateM`: This function creates multiple instances of a hardware module.

- The **inner `replicateM(mk_MAC_top)`** creates a row of 4 MAC units.
- The **outer `replicateM`** creates 4 such rows to form the 4x4 grid.

b. **`Vector#(4, FIFO#(Bit#(16))) fifo_A_rows <- replicateM(mkFIFO);`**

- Each row of the systolic array has its own FIFO buffer to store incoming AAA values.
  - `Vector#(4, FIFO#(Bit#(16)))`: A vector of 4 FIFOs, one for each row.
  - `FIFO#(Bit#(16))`: Each FIFO can store 16-bit values (corresponding to the width of A).
  - `replicateM(mkFIFO)`: Creates 4 instances of FIFO buffers.
- The FIFOs ensure that A values are provided to each row **sequentially and synchronously** as required by the systolic array.
- The first column of the systolic array in each row retrieves A values directly from these FIFOs.
- A values are propagated horizontally across the row of MAC units after being processed by each unit.

c. **`Vector#(4, FIFO#(Bit#(16))) fifo_B_cols <- replicateM(mkFIFO);`**

- Each column has a dedicated FIFO to store B values.
- `FIFO#(Bit#(16))`: Each FIFO stores 16-bit values for B.
- The first row of the systolic array retrieves B values directly from these FIFOs.
- B values flow vertically down the column, passed between MAC units.

d. **`Vector#(4, FIFO#(Bit#(1))) fifo_select_cols <- replicateM(mkFIFO);`**

- These FIFOs store **select signals** that control the behavior of the MAC units in each column.
  - `FIFO#(Bit#(1))`: Each FIFO holds 1-bit control signals.
  - The select signals determine if a MAC operation is active or skipped for a given column.
- Select signals flow vertically down the column alongside the BBB values.

e. `Vector#(4, FIFO#(Bit#(32))) fifo_C_cols <- replicateM(mkFIFO);`

- These FIFOs store **initial C values**, often used as bias or initial accumulation values in matrix operations.
- `FIFO#(Bit#(32))`: Each FIFO holds 32-bit values for C.
- The first row of the systolic array retrieves C values directly from these FIFOs.
- C values are used in the first MAC operation and are propagated vertically down the column as the accumulation progresses.

f. Registers to store MAC results

```
Vector#(4, Reg#(Bit#(32))) mac_results <- replicateM(mkReg(0));
```

- These registers store the final results of the systolic array computations, one for each column.
    - `Vector#(4, Reg#(Bit#(32)))`: A vector of 4 registers, one for each column.
    - `Reg#(Bit#(32))`: Each register stores a 32-bit result.
    - `mkReg(0)`: Initializes each register to 0.
    - The last row of the systolic array writes its output to these registers.
    - At the end of the computation, the `get_MAC_results` method reads these registers to retrieve the results.

g. Rules

The **systolic_step** rule defines the core operational logic of the 4x4 systolic array, orchestrating data flow and computation across all the processing elements (MAC units) in the array. This rule is executed as a single atomic step, ensuring that all data processing within the systolic array happens synchronously and deterministically.

The **systolic_step** rule:

- Handles the **input flow** of AAA, BBB, and CCC values into the array.
- Propagates data through the **grid of MAC units**.
- Performs **MAC operations** in each cell.
- Accumulates and stores the **final results** in the last row.

Data flow in the array:

- A values flow **horizontally** across rows.
- B values flow **vertically** down columns.
- C values propagate from the top to the bottom row, accumulating results

Nested loops are used to process every MAC unit in the 4X4 grid. Outerloop (i) iterates through the rows (0 to 3), and the inner loop (j) iterates through the columns (0 to 3).

The logic used for handling the flow of A value is:

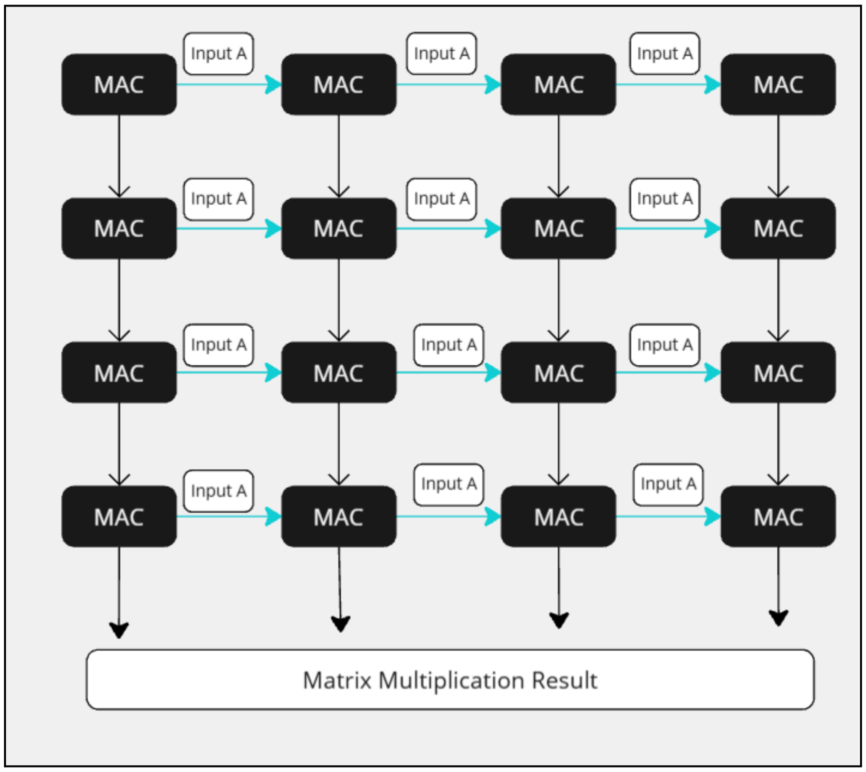


Fig1: The flow of data values A across the array

First Column (`j == 0`):

- Fetch A value from the input FIFO (`fifo_A_rows[i]`) for row iii.
- Dequeue the FIFO (`deq()`) after reading the value.
- Pass AAA to the first MAC unit in the row using `put_A(a)`.

Subsequent Columns (`j > 0`):

- Fetch A from the **output of the MAC unit in the previous column** using `get_A_out()`.
- Pass it to the current MAC unit using `put_A(a)`.

Similarly, the logic used for Handling B values are:

- **First Row (`i == 0`):**
  - Fetch B and select values from the input FIFOs (`fifo_B_cols[j]` and `fifo_select_cols[j]`).
  - Dequeue the FIFOs after reading the values.
  - Pass B and select values to the first MAC unit in the column using `put_B(b)` and `put_select(s)`

- **Subsequent Rows (`i > 0`):**
    - Fetch B and select values from the **output of the MAC unit in the previous row** using `get_B_out()` and `get_select_out()`.
    - Pass them to the current MAC unit.

C values are the bias or initial accumulation. They are properly passed to each MAC unit using the logic,

- **First Row (`i == 0`):**
    - Fetch C value from the input FIFO (`fifo_C_cols[j]`).
    - Dequeue the FIFO after reading the value.
    - Pass C to the first MAC unit in the column using `put_C(c)`.
- **Subsequent Rows (`i > 0`):**
    - Fetch the accumulated result (MAC) from the **MAC unit in the previous row** using `get_MAC()`.
    - Pass this as the new C value to the current MAC unit using `put_C()`

The final results are stored in the output registers for each column:

- Only the **last row (i == 3)** writes its results to `mac_results`.
- The `get_MAC()` method retrieves the final accumulated result from the MAC unit.

Data Flow	**Direction**	**Input Source**	**Propagation**	**Final Destination**
A Values	Horizontal (→)	**fifo_A_rows[i]**	From left to right in each row	Each MAC unit in a row
B Values	Vertical (↓)	**fifo_B_cols[i]**	From top to bottom in each column	Each MAC unit in a column
Select Signals	Vertical (↓)	**fifo_select_cols[i]**	From top to bottom in each column	Each MAC unit in a column
C Values	Vertical (↓)	**fifo_C_cols[i]**	From top row or prior row's MAC result	MAC units in each column
Final Results	-	Stored in Registers	Computed by last row's MAC units	**mac_results[j]** for each column

MAC Top Module design

The updated **MAC (Multiply-Accumulate) Top Module** is designed to be used more effectively as a systolic array's processing elements. The module integrates input handling, computation, and output routing. This design incorporates additional FIFOs and decoupled input/output pathways to support the data flow across the systolic array.

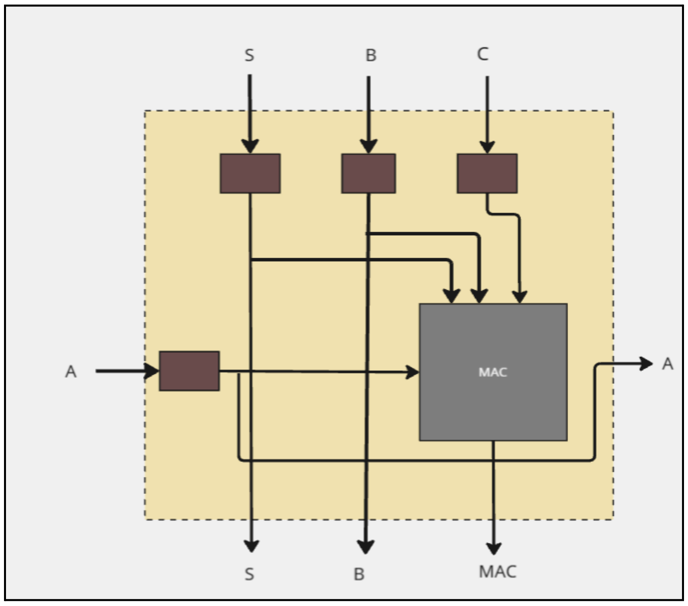


Fig: New Top module for MAC.

FIFOs are used for input handling. The module introduces separate FIFOs for buffering incoming data.

- **FIFO for A:** Buffers the horizontal data flow for matrix A..
- **FIFO for B:** Buffers the vertical data flow for matrix B.
- **FIFO for C:** Buffers the accumulation/bias values (C).
- **FIFO for Select Signals:** Buffers the select signals (S) controlling the operation mode of the MAC unit.

Separate FIFOs are added for A, B, and select signals for **output routing**:

- Output FIFOs pass the propagated values (A,B, and S) to the next MAC unit in the systolic array.
- This allows seamless horizontal and vertical data flow

The core MAC computation is performed by the `MAC_ifc` interface, instantiated as `mac.` The module defines a single rule, process_inputs, to process the buffered inputs:

- Fetch A,B, C, and S values from the respective FIFOs.
- Forward these values to the core MAC unit for computation.
- A, B, and select signals are enqueued into output FIFOs and passed to neighboring MAC units.
- The results from the core MAC unit are directly available via `get_MAC`.

The select signal (S) allows the module to operate in different modes:

- S=1 is for int32 computation
- S=0 is for float32 computation

Verification of Design

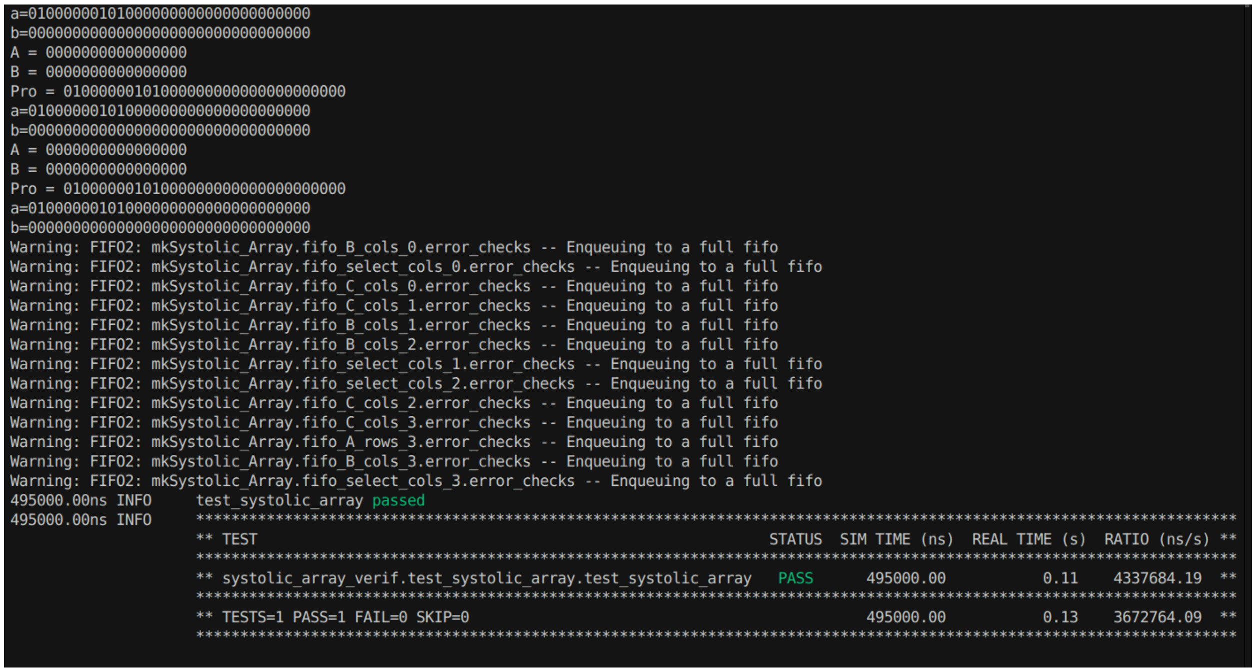
Test Model

The `model_systolic` function is a Python-based reference model designed to verify the behavior of the systolic array module implemented in BSV. It accepts three operands (`a`, `b`, `c`) and a `select` signal, simulating both integer and floating-point MAC operations based on the value of `select`. When `select` is `0`, it performs integer multiplication and addition, while a value of `1` triggers floating-point arithmetic. The function classifies the inputs as 'integer', 'fractional', 'negative', or 'other' for coverage analysis, ensuring comprehensive testing across different scenarios. The computed result is returned and logged, facilitating direct comparison with the BSV MAC module's output. This model plays a crucial role in validating the functionality of the hardware implementation and identifying discrepancies during testing, making it an integral part of the verification process.

The Python test model test_model.py leverages Cocotb to simulate, control, and verify a hardware systolic array module. It performs the following key tasks:

1. Initializes the clock and resets the MAC module.
2. Loads inputs from files for integer and floating-point testing.
3. Tests integer matrix multiplication by setting the select signal to 0, processing the inputs, and comparing the results with the reference model.
4. Tests floating-point matrix multiplication operation with select set to 1.
5. Uses assertions to validate outputs, ensuring the hardware implementation matches the expected behavior.
6. Exports coverage data for further analysis.

RUN STATUS:



```
a=01000000101000000000000000000000
b=00000000000000000000000000000000
A = 0000000000000000
B = 0000000000000000
Pro = 01000000101000000000000000000000
a=01000000101000000000000000000000
b=00000000000000000000000000000000
A = 0000000000000000
B = 0000000000000000
Pro = 01000000101000000000000000000000
a=01000000101000000000000000000000
b=00000000000000000000000000000000
Warning: FIFO2: mkSystolic_Array.fifo_B_cols_0.error_checks -- Enqueuing to a full fifo
Warning: FIFO2: mkSystolic_Array.fifo_select_cols_0.error_checks -- Enqueuing to a full fifo
Warning: FIFO2: mkSystolic_Array.fifo_C_cols_0.error_checks -- Enqueuing to a full fifo
Warning: FIFO2: mkSystolic_Array.fifo_C_cols_1.error_checks -- Enqueuing to a full fifo
Warning: FIFO2: mkSystolic_Array.fifo_B_cols_1.error_checks -- Enqueuing to a full fifo
Warning: FIFO2: mkSystolic_Array.fifo_B_cols_2.error_checks -- Enqueuing to a full fifo
Warning: FIFO2: mkSystolic_Array.fifo_select_cols_1.error_checks -- Enqueuing to a full fifo
Warning: FIFO2: mkSystolic_Array.fifo_select_cols_2.error_checks -- Enqueuing to a full fifo
Warning: FIFO2: mkSystolic_Array.fifo_C_cols_2.error_checks -- Enqueuing to a full fifo
Warning: FIFO2: mkSystolic_Array.fifo_C_cols_3.error_checks -- Enqueuing to a full fifo
Warning: FIFO2: mkSystolic_Array.fifo_A_rows_3.error_checks -- Enqueuing to a full fifo
Warning: FIFO2: mkSystolic_Array.fifo_B_cols_3.error_checks -- Enqueuing to a full fifo
Warning: FIFO2: mkSystolic_Array.fifo_select_cols_3.error_checks -- Enqueuing to a full fifo
495000.00ns INFO     test_systolic_array passed
495000.00ns INFO     ***********************************************************************************************************************
                     ** TEST                                                          STATUS  SIM TIME (ns)  REAL TIME (s)  RATIO (ns/s) **
                     ***********************************************************************************************************************
                     ** systolic_array_verif.test_systolic_array.test_systolic_array   PASS      495000.00           0.11    4337684.19  **
                     ***********************************************************************************************************************
                     ** TESTS=1 PASS=1 FAIL=0 SKIP=0                                             495000.00           0.13    3672764.09  **
                     ***********************************************************************************************************************
```