Functional Instruction Set Simulator (ISS) of a Neural Network Accelerator IP with native "brain float16" format

Debajyoti Mukherjee	Arpita Sahu	Arathy B S	Saranga P Pogula
Pre-Si Verif Lead	Pre-Si Verif Engineer	Pre-Si Verif Engineer	Principal Engineer
Intel Corporation, India	Intel Corporation, India	Intel Corporation,India	Intel Corporation, India
debajyoti.mukherjee@intel.com	arpita.sahu@intel.com	arathy.b.s@intel.com	saranga.p.pogula@intel.com

***Abstract*- Functional models are crucial for demonstrating the functional correctness of Neural Network compute accelerator IPs. Neural network workloads are based on the brain floating point (bfloat16) data type. Proposed functional model incorporates a native bfloat16 generator to address the incompatibility between gcc / g++ compilers and brain float datatypes. Moreover, working with big GEMM (General Matrix Multiplication) or SpMM (Sparse Matrix Multiplication) Work Loads (Dense or Sparse) and debugging the failures related to data integrity is incredibly difficult. This paper addresses the quality and TTM (Time to Market) challenge of complex neural network accelerator design by proposing a functional model-based scoreboard (for hardware debug) or software model (ISS as a virtual model for software developers) using SystemC.**

I. Introduction

A CISC (Complex Instruction Set Computer) ISA (Instruction Set Architecture) program can be extremely complex to debug as a hardware implementation. The proposed Functional Model (FM) executes the assembly code based on ISA of processor IP, decodes all instruction, and executes as expected to be done by the DUT. With the said model, there would be greater visibility and debugging capabilities for the DUT bring up in micro steps of execution.

- **The proposed model is developed in SystemC using High level synthesis** (HLS) flow.
- ISA related **architectural changes can be accommodated easily** in the FM.
- FM is **golden and pre-validated** separately, so it **eases RTL / SystemC debugging** as it **has all the internal debug information/details of the Instruction Processing**.
- FM is the **source of Debug Trackers**. Trackers are very helpful for deep internal Data Mismatch debugs.
- **Validates the NN Compiler Generated program HEX code** – an additional advantage, sanitizing the compiler/assembly flow as well.

II. Description

For a Neural Network workload, we have designed the compiler output in a JavaScript Object Notation (JSON) format which is then parsed to generate an associated program HEX code. The HEX code then gets fed as an input to both FM and synthesizable design IP. Depending upon the design, the environment takes up either the SystemC FM or its SV variant.

- In SV FM, the C++ pseudo memory is accessed with the help of SV DPI function calls.
- Both the FM and the synthesizable design gets instantiated inside the same testbench (TB) where they receive stimulus for preloading the data in a shadow memory and configuring the Control Status Registers (CSR) for their functioning.
- FM dumps out partially processed data at every stage of the design and thus acts as a source of debug trackers. Both the DUT and FM completes processing a given WL as two different parallel threads and their outputs are compared in a post processing step.

FM played a crucial role in identifying many issues in the compiler operation very early in the design cycle which would have been otherwise caught only at a later part. Also, the FM trackers eased the data integrity debugs and fastened the RTL bring up. Thus, beyond a golden reference model, FM helped improve the TTM of the ip. The following figure describes single cluster verification environment.

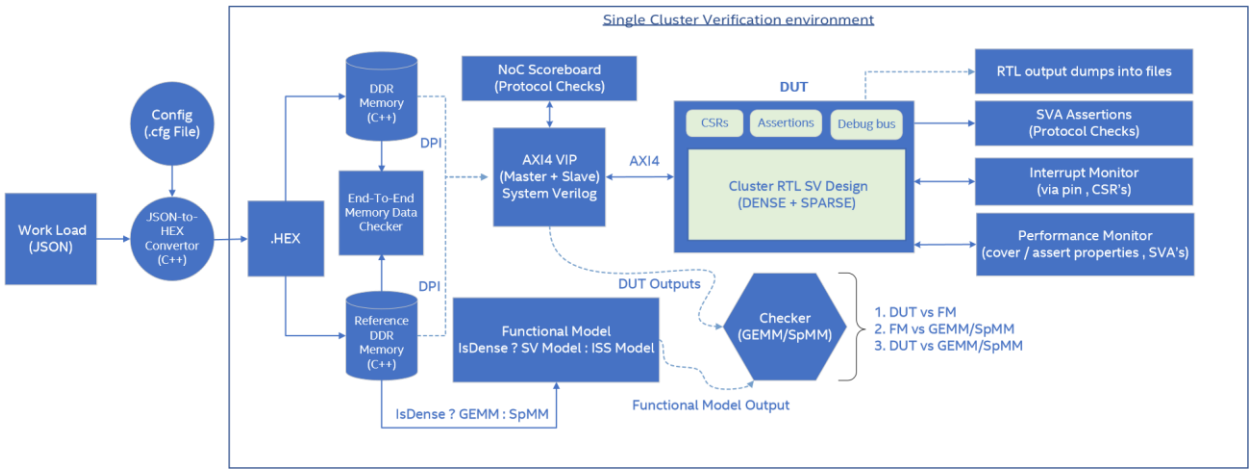

Fig. 1 Single Cluster Verification Environment

III. BFLOAT16 MODELING

The proposed FM environment makes use of brain floating-point format (bfloat16 or BF16) data precision with the help of a native BF-16 generator integrated to the environment. The native BF-16 generator is a union-based construct and consists of APIs that performs the required math operation in float32 (FP32) precision and represents the results in BF-16. This exploits the ease of conversion between BF-16 and FP32 datatype formats thereby overcoming the absence of a C++ brain float16 compiler.

IV. FUNCTIONAL MODEL STRUCTURE

FM possess a header file which defines the data structures that tackles architectural breakdown of bigger workload data into various smaller granularities and thereby used in synchronization purpose across concurrent threads where individual threads interact among each other using Transaction Level Modeling (TLM). The four concurrent threads which majorly constitutes FM functionality are IFETCH, LOAD, COMPUTE and STORE.

A. iFETCH Thread

iFETCH (or Instruction Decode) thread gets triggered when a specific CSR write happens from the FM TB. iFETCH collects the program start address and program size from the CSRs which are loaded by the FM environment. iFETCH grabs the instructions in chunks of Maximum Read Request Size and processes them accordingly. It decodes the compiler generated program HEX file in an instruction-by-instruction manner and pushes the instructions to corresponding TLM queues for further consumption by LOAD/COMPUTE/STORE threads. Also, iFETCH maintains a track of all byte availabilities for correct ordering of data processing.

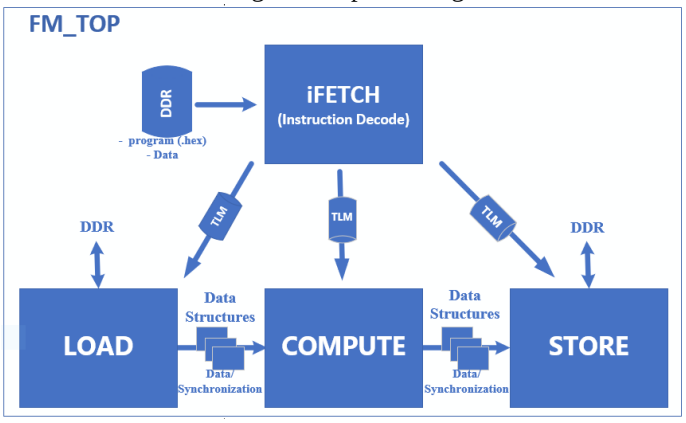

Fig. 2 Functional Model Top Module

B. Load Thread

LOAD thread perceives a load instruction once the iFETCH pushes an entry in the respective queue. Associative Arrays using C++ map constructs are used inside the FM to mimic SRAM like behavior. Each LOAD contains information about the size of data to be fetched from memory, DRAM start address and a flag bit to indicate whether the instruction belongs to A-Matrix (Processing-Element-Matrix) or B-Matrix (Weight-Matrix). Thus, LOAD thread fetches the data present in the DRAM into corresponding associative arrays as shown below and waits for the next available instruction in the TLM queue.

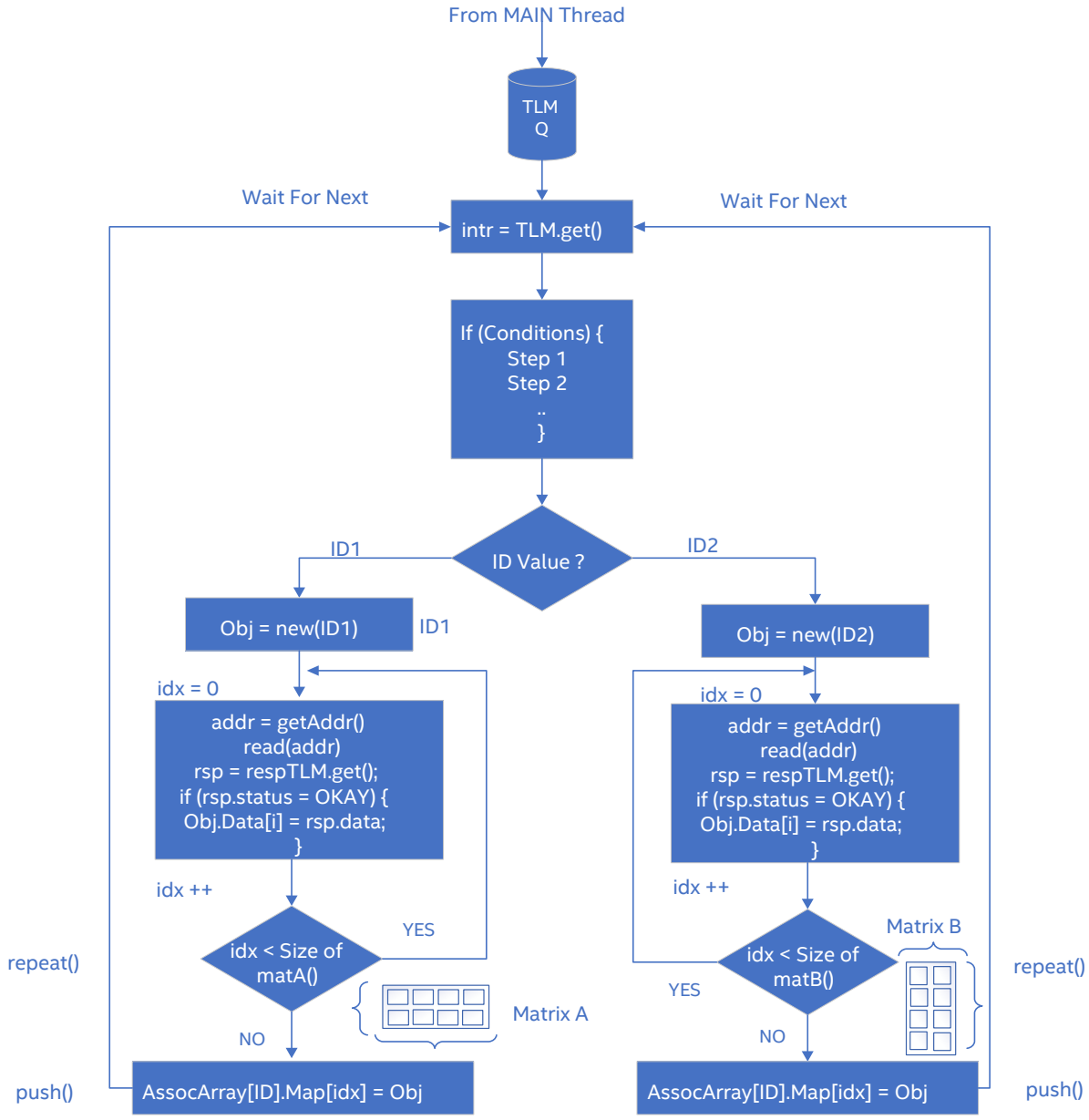

Fig. 3 Functional Model LOAD thread

C. Compute Thread

COMPUTE thread has a dependency on LOAD. Once the LOAD thread completes its execution, it communicates the same to COMPUTE thread via SystemC semaphores. A COMPUTE instruction typically holds information on the granular data to be fetched from the associative arrays to perform Multiplication and Accumulation operation (MAC). The resultant MAC data is also kept inside a temporary buffer for partial processing's. COMPUTE requires specific operands to be made available for its operation and thereby keeps a track of the entire Matrix multiplication. Once the result of the MAC compute doesn't need any further accumulation on the top of existing partials, a new entry gets created inside the associate array storage element to store the final computed value. Compiler sets a 'DONE' flag bit when a given COMPUTE marks the completion of computation in a common direction. Based upon the DONE Flag, COMPUTE sends out a note to STORE thread to move the temporary final computed values (stored inside Associative Array-C) back to DRAM.

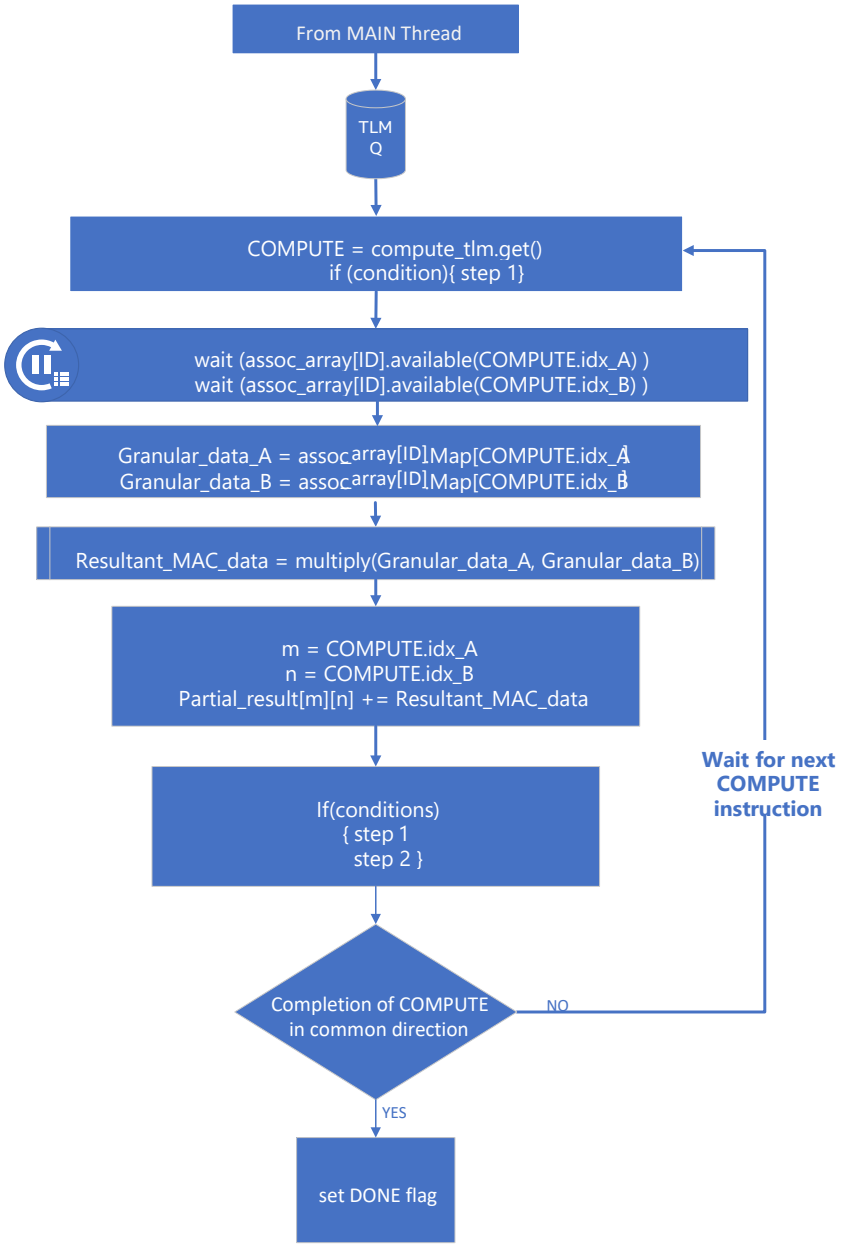

Fig. 4 Functional Model COMPUTE thread

D. Store Thread

STORE maintains a dependency with COMPUTE as per as ordering of processing is concerned. STORE instruction holds the index of the entry which is to be fetched from the C Matrix Array, and the final DRAM destination address. Once the STORE gets completed, the given entry gets deleted from the temporary associative array. When FM finishes the processing of all STORE instructions, it marks the completion of the workload. At this point, different CSRs get set from FM_TOP module for end of workload detection. Upon completion the storage elements in all the FM threads goes back to reset to take up another workload.

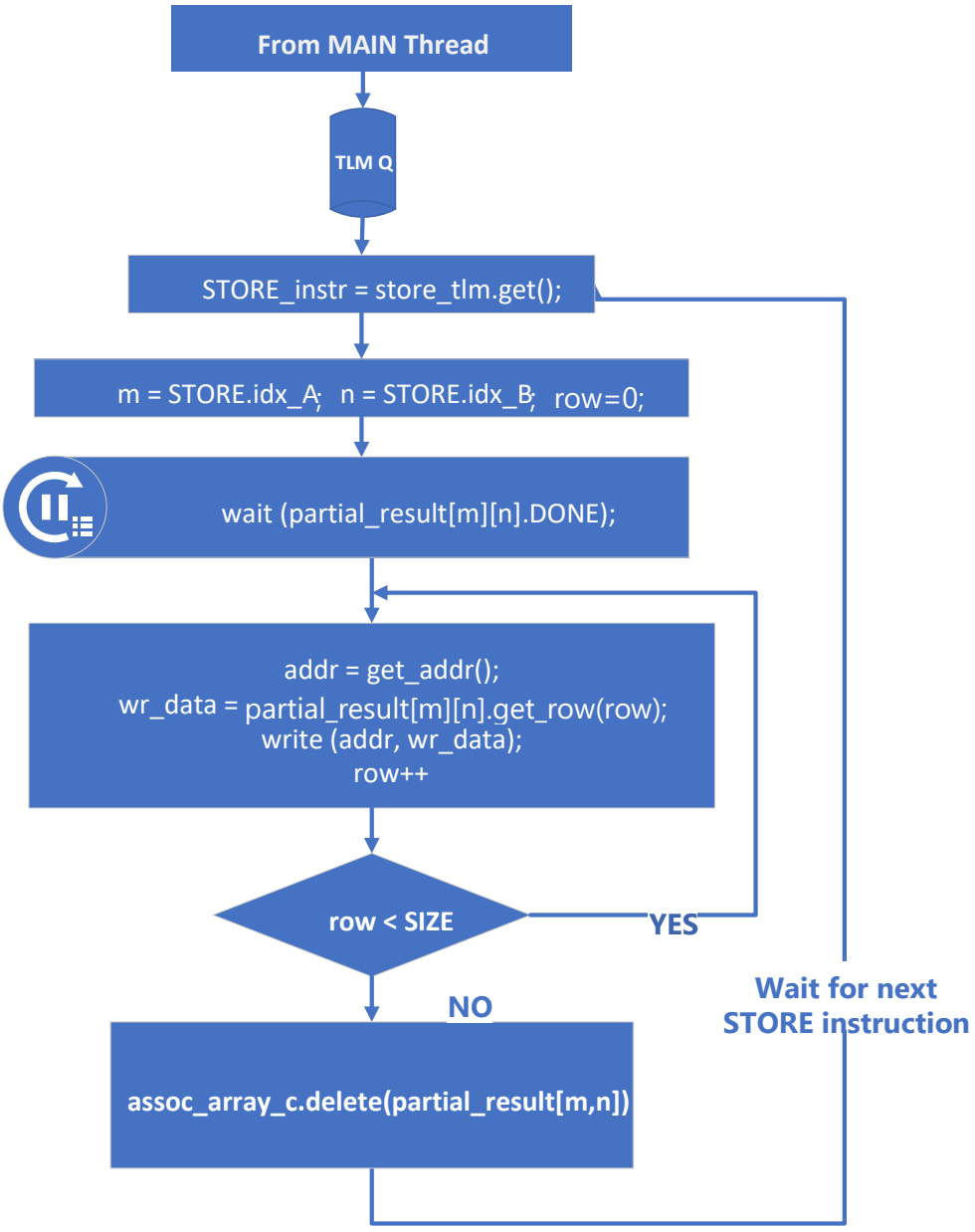

Fig. 5 Functional Model STORE thread

V. RESULTS

- Verified 95% Functional Correctness of IP Design prior to FPGA prototyping
- Validated the Neural Network IP ISA (Instruction Set Architecture) very early in the project, giving feedback to architecture definition
- Done early verification of Compiler operation, finding several issues in compiler architecture as well
- Very early freezing of collateral (.HEX / disassembly) file formats
- Used as verification scoreboard for both HLS Design (SystemC) and Generated RTL (System Verilog)
- Enabled data integrity checks in pre-silicon verification environment, which were also reused at the FPGA Platform
- Solution provides flexibility to the end user to pick either SystemC or its System Verilog variant, while reusing C++ memory and checkers using SV DPI feature of System Verilog
- FM code has gone through Profiling and has got improvisations for simulation throughput optimizations.
- FM is also integrated with end-to-end GEMM checker (simple high level C code) for Dense Matrix multiplication Operations. GEMM checker is a generic matrix multiplication algorithm which has a computational complexity on the order of n^3 to multiply two n x n matrices.
- Trackers and Debug Hooks are available in FM for ease of use, which enabled faster RTL1.0 design freeze with highest quality prior to FPGA and for SOC