UNIVERSITY OF CALIFORNIA

Santa Barbara

Hardware implementation and analysis of memory interfaces to integrate a vector accelerator into a manycore Network-on-Chip

A Thesis submitted in partial satisfaction of the

requirements for the degree Master of Science

in Electrical and Computer Engineering

by

Ci-Chian Lu

Committee in charge:

Professor Jonathan Balkind, Chair

Professor Timothy Sherwood

Professor Yufei Ding

June 2023

The thesis of Ci-Chian Lu is approved.

Yu-Fei Ding

Timothy Sherwood

Jonathan Balkind, Committee Chair

June 2023

ACKNOWLEDGEMENTS

I would like to express my gratitude to my advisor, Professor Jonathan Balkind, for his tremendous support, patience, and encouragement throughout my project. He provided guidance during challenging moments, and I am truly thankful for his unwavering belief in my abilities. I also want to extend my thanks to my committee members, Professor Yu-Fei Ding and Professor Timothy Sherwood, for their valuable feedback and suggestions that significantly improved the quality of my work. I am deeply appreciative of the assistance provided by Dr. Mohammad Hossein Askari Hemmat from Polytechnique Montréal, whose contributions were instrumental in the successful completion of my project. I am also grateful to Matteo Perotti from ETH Zurich for his helpful responses to my questions and suggestions regarding my implementations.

Lastly, I want to express my heartfelt appreciation to my parents for their financial support, which made it possible for me to embark on a two-year journey at UCSB.

ABSTRACT

Hardware implementation and analysis of memory interfaces to integrate a vector accelerator into a manycore Network-on-Chip

by

Ci-Chian Lu

In recent years, there has been a growing demand for vector processors due to their increasing application in deep-learning applications. On the other hand, with the strong need for energy efficiency and high performance, heterogeneous architecture plays an important role and becomes increasingly complex. However, the way of connecting the memory hierarchy to the vector processor in SOC (System-on-Chip) is critical to the system's performance [8]. This work presents tile design which is based on OpenPiton and BYOC [4] [3]. Tile consists of a 64-bit, single-issue, in-order RISC-V core Ariane [14], along with a 64-bit vector processor ARA [7] [13] which implemented RISC-V V extension version 1.0. This work makes the following contributions. First, it involves the design and implementation of an adapter (bridge) that converts memory request from AMBA AXI to OpenPiton NoC. This adapter enables ARA memory access functionality and facilitates the integration of future accelerators into OpenPiton. Secondly, a tile design is presented, which includes ARA, a RISC-V vector processor, Ariane (a RISC-V core), L1.5 cache, L2 cache, and the implemented bridge. The performance of the tile is evaluated using different versions of bridges connected to the last-level cache (LLC) or off-chip memory. The analysis indicates that a wide data width bridge does not necessarily improve performance significantly. Several factors, such as NoC traffic confliction or unused data fetch, can narrow the performance gap between small and large width bridges. Furthermore, the experiments demonstrate that memory exhibits advantages when dealing with large data widths, and memory saturation also occurs during LLC access. Finally, the thesis proposes the implementation of MSHR (Miss Status Handling Register) and extends this design to manycore architectures to enhance performance.

TABLE OF CONTENTS

I. Introduction

With the growing demand for enhanced performance and the end of Dennard scaling, the limitations of processor operation frequency and the requirement for energy efficiency cannot be adequately addressed by expensive and complex general-purpose processors alone. Consequently, alternative architectures have been proposed to augment computational capabilities. These include manycore architectures, which leverage instruction-level parallelism [9], and single-instruction multiple data (SIMD) architectures, which exploit data-level parallelism.

Vector processors are revamped based on the SIMD design paradigm. Vector processors offer multiple advantages when dealing with regular data parallelism. A single instruction can fetch a long vector, amortizing the instruction, fetching ***overhead,*** and controlling logic complexity cost. However, the disadvantage of vector machines becomes apparent when the data parallelism is not regular, requiring heavy software code rewriting [10].

Vector processors could enhance the performance efficiently with high data parallelism program, however, due to the high data volume the vector needs, memory access latency becomes one of the bottlenecks for integrating vector machine. As a result, the way of combining the vector core also matters. In [8], they proposed three ways to connect accelerators, and two of them - loosely out-of-core coupling with Direct Memory Access to the Last-Level Cache (LLC-DMA) and loosely out-of-core coupling with Direct Memory Access to DRAM (DRAM-DMA) - are similar to my architecture. Their work shows that the LLC-DMA performs better due to the bandwidth bottleneck of off-chip DRAM access, and the cache pollution caused by unnecessary data stored in the cache plays a minor role in this comparison. The low access latency and high bandwidth mitigate the cache pollution.

This work presents a tiled-based processor based on OpenPiton, a heterogeneous manycore processor research platform [3] [4]. I focus on the hardware implementation and design of the memory access interface of vector processor and minimizing the need to rewrite the software code to assess the influence of the combination of vector machine and OpenPiton P-MESH memory system maximally.

This work has the following contributions. First, analysis of different-sized bridges: The study examines bridges of varying sizes and their impact on system performance. While the 128-bit bridge theoretically provides higher bandwidth, it is important to consider the frequency of cache miss and memory access pattern. The work compares the connection of the bridge to the LLC and off-chip memory. Connecting the bridge to off-chip memory introduces higher latency and potential memory saturation issues [8]. However, the large capacity of memory may take advantage on the high data demand program. Finally, the work proposes future improvements, including the introduction of a Modified Store Hit Request (MSHR) tags block design to enhance throughput by facilitating out-of-order data accessing. Additionally, the extensibility of the OpenPiton platform suggests the potential implementation of a manycore architecture. These proposals outline avenues for further enhancing system performance and scalability.

II. Background

A. SIMD vs MIMD

In computer architecture, two fundamental paradigms exist for exploiting data-level parallelism: SIMD (Single Instruction, Multiple Data) and MIMD (Multiple Instruction, Multiple Data). SIMD architecture allows multiple processing units or functional units to execute the same instruction simultaneously on different data streams. The results are then stored in the same memory location. SIMD architecture efficiently exploits data-level parallelism, as each instruction can simultaneously be applied to multiple data elements. However, the grain size must be sufficiently large to utilize the parallelism fully. One common variation of SIMD is the vector architecture, where the processor operates on the same vector or data set in consecutive cycles.

On the other hand, MIMD architecture allows multiple processing units to execute different instructions on different data streams concurrently. Therefore, while MIMD architectures also exploit data-level parallelism, they typically incur higher overhead than SIMD architectures. The overhead refers to the additional complexities and coordination required to execute different instructions on different data streams simultaneously. Regarding energy efficiency, SIMD architectures have an advantage over MIMD architectures since they amortize the need for fetching instructions. SIMD architectures can reduce the energy consumption associated with instruction fetching and decoding by executing the same instruction on multiple data elements [11].

B. OpenPiton Introduction

OpenPiton is a platform designed for developing, simulating, FPGA emulation, and constructing many-core processors and system-on-chip (SoC) architectures [4]. It utilizes a tiled many-core approach, featuring a 64-bit architecture and a distributed, directory-based cache coherence system implemented across three physical 2D mesh Network-on-Chip (NoC) layers.

The architecture of OpenPiton can be divided into two components: Intra-chip and Inter-chip. In the Intra-chip component, the tiles within each chip are connected in a 2D mesh topology using NoC channels. This facilitates efficient communication and data transfer between the tiles, enabling parallel processing within the chip.

The Inter-chip component focuses on the connectivity between the tiled array and off-chip logic. OpenPiton includes an off-chip interface that enables integration with external components, such as off-chip DRAM or IO devices. This allows for efficient data exchange between the manycore processor and external resources, enhancing the overall system capabilities.

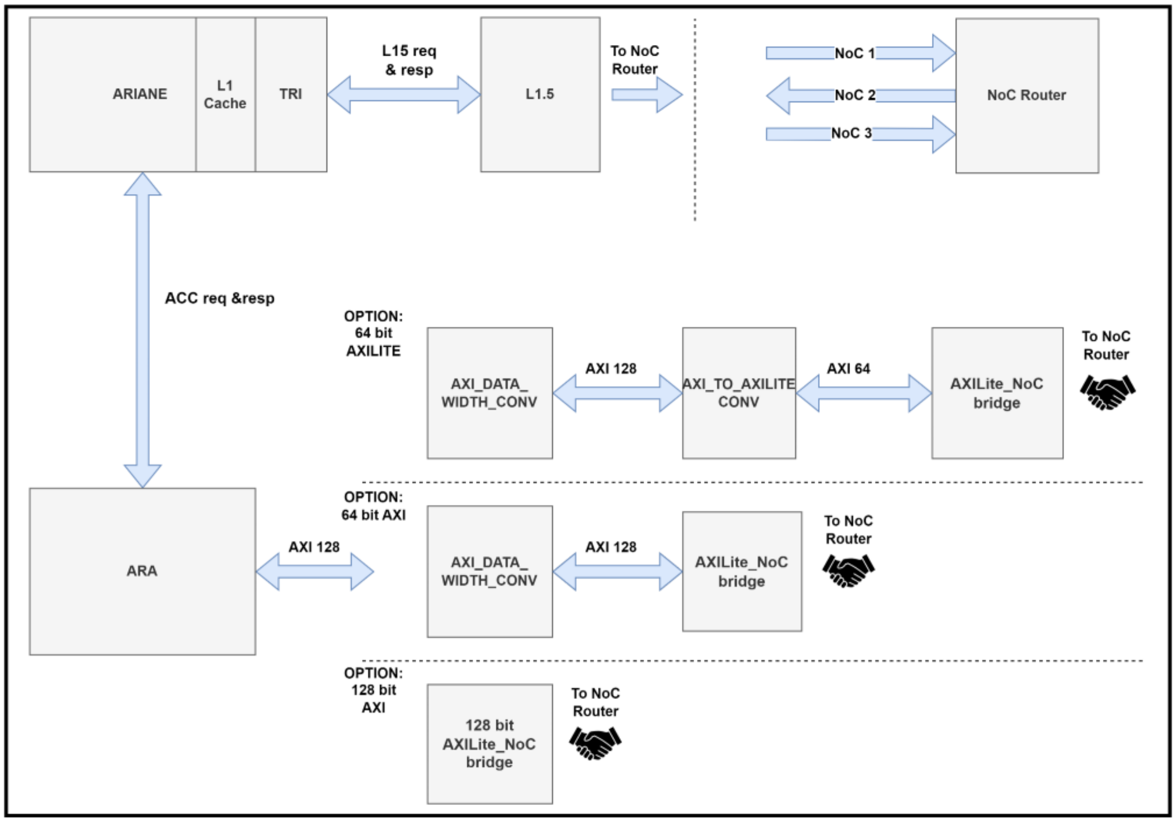


Figure 1. Tile Architecture, this work presents three different types of bridges, AXI-Lite bridge, AXI-64 bridge, and AXI-128 bridge.

III. Tile Architecture

This thesis presents a 2 x 2 mesh manycore architecture, where each mesh node consists of a tile. The architecture of the tile is depicted in Figure 1. It includes an Ariane core, a 64-bit RISC-V in-order processor, and an ARA core, a 64-bit RISC-V vector processor implementing the RVV extension 1.0. The P-Mesh collective system, as described in [3], is utilized to address cache coherence requirements. This system incorporates a two-level cache hierarchy comprising L1.5 and L2 caches, interconnected through three NoC channels. Specifically, the L1 cache in the Ariane core is connected to the L1.5 cache via the Transaction Response Interface (TRI). The TRI facilitates efficient cache coherence and communication between the L1 cache and the P-Mesh system. In addition, it supports bidirectional communication with a handshake mechanism, allowing the core to send requests (e.g., loads, stores, invalidations) and receive corresponding responses from the P-Mesh system. In contrast, the ARA vector processor follows a different integration approach than the Ariane core. It initiates memory requests using the AXI with a 128-bit data width and 64-bit address width. To accommodate various bridge sizes and support the AXI-Lite protocol, the data downsizer and the AXI to AXI-Lite converter from [12] are employed. These components facilitate data width conversion from 128 to 64 bits and enable seamless protocol transition.

A bridge component is implemented and included in the design. It converts the AXI or AXI-Lite packets from the core side to OpenPiton NoC packets, which are then transmitted to the P-Mesh system (LLC) to support ARA's load or store requests. The bridge's detailed design is also described in this thesis.

A. OpenPiton + Ariane

Ariane [14] is an open-source, single-issued, six-stage, in-order RISC-V 64-bit CPU. The six stages include program counter (PC) generation, instruction fetch, instruction decode, issue stage, execute stage, and commit stage. OpenPiton, along with the Ariane core, is a verified integrated design described in [3]. However, as outlined in the ARA paper, some architectural changes are required when integrating ARA. First, Ariane lightly decodes the instruction to determine if it is a vector instruction. If it is, the instruction is dispatched when it reaches the top of the scoreboard, which logically sits between Ariane's issue stage and execution stage.

Cache coherence is maintained in the BYOC implementation [3], which includes the L2 cache and the BYOC private cache (L1.5 cache). In the original ARA design, the L1 cache was extended with a write-through policy to invalidate the corresponding cache line, ensuring that the data in memory accessed by ARA is always up-to-date. In OpenPiton, the LLC (Last-Level Cache) tracks the states of all cache lines. To minimize NoC (Network-on-Chip) traffic between the BPC (BYOC Private Cache) and LLC, only the L1 cache in Ariane employs a write-through policy, while the BPC and LLC utilize a write-back policy. Cache coherence is maintained through invalidations sent from the L2 cache to the L1.5 cache. These design considerations and modifications ensure the integration of the ARA core into the OpenPiton architecture while maintaining cache coherence and optimizing the overall system's performance.

B. ARA architecture and its load/store unit

ARA [13] is a 64-bit vector processor that implements RISC-V vector extension version 1.0. It acts as a coprocessor alongside Ariane. ARA can be configured with a variable number of identical lanes, each lane having a portion of the complete ARA's vector register file (VRF). The VRF consists of eight 1RW port banks and execution units, such as integer multipliers and the FPU. In this particular design, the number of lanes is set to four. Inter-lane communication primarily relies on the VLSU (Vector Load/Store Unit) and SLDU (Slide Unit). For instance, when executing an instruction that requires access to all banks, such as inserting an element into a vector, the VLSU and SLDU facilitate the necessary communication. The length of the vector registers is also configurable, representing the number of elements in a vector register. For this work, the vector length is set to 4096.
The dispatcher serves as the interface between ARA and Ariane. It receives vector instructions no longer speculative from Ariane and dispatches them to the accelerator. Finally, the sequencer is crucial in tracking the instruction status, dispatching instructions to different execution units, and providing acknowledgments to Ariane.
The VLSU, which is tightly connected to the bridge presented in this work, comprises units such as the address generator, load unit, and store unit. The address generator is responsible for determining the memory address to be accessed. It incorporates a finite state machine with three implemented states. First, the generator awaits a vector load/store instruction in the *IDLE* state. In the *ADDRGEN* state, the generator generates a series of AXI requests for the load or storage units. Notably, in the *ADDRGEN_IDX_OP* state, the generator generates a series of AXI requests while reading a vector of offsets from the lanes. This is particularly useful for scatter/gather operations. When the address generator receives a memory request from the

main sequencer, it checks for address misalignment with specific width requirements. Additionally, if the request corresponds to a unit-stride load/store, meaning it accesses contiguous memory addresses, it is transformed into an AXI burst request.

C. AXI and AXI-Lite Protocol

The AMBA AXI protocol [1] is introduced in the design to provide high bandwidth and low latency capabilities. It offers several key features, including separate address and data phases, burst-based transactions, support for misaligned transactions, and separate write and read channels. Compared to its predecessor, the AMBA Advanced High-performance Bus (AHB) protocol, AXI provides more flexible data control and transmission options. For instance, AXI supports misaligned transactions, allowing the address to be misaligned with the AXI data width. It also supports multiple outstanding transactions, enabling a master to send multiple read or write transactions before receiving the response from the slave.
The AXI protocol consists of five channels: the write address channel (AW channel), write data channel (W channel), write response channel (B channel), read address channel (AR channel), and read data channel (R channel). Each channel operates independently, facilitating the high bandwidth of the AXI protocol. All channels use the VALID-READY handshake signal to process address, data, and control signals.

Here, I describe the signals implemented in the design:

(1) AW/AR Channel: The master initiates transactions by driving the start address of each transaction and the relevant control signals.

(2) Write/Read Address: This signal indicates the start address of each transaction. Burst Length: This signal indicates the number of transfers in a transaction.

(3) Burst Size: This signal indicates the maximum number of bytes in each data transfer. Burst Type: AXI supports three burst types - FIXED, INCREMENT, and WRAP. The critical difference lies in how the address changes in each transfer. In the FIXED type, the address remains the same for each transfer. In the INCREMENT type, the address increments by the transaction size for each transfer. In the WRAP type, the address wraps around the lower address when the upper address is reached. The equation determines the wrap boundary: the size of the transaction multiplied by the number of transfers in the burst.

   The WRAP type is commonly used in cache line access to improve performance. For example, if a specific byte in a cache line needs to be loaded first by the CPU, the start address can be set to the address of that particular byte, allowing it to be sent back to the CPU more efficiently.

(4) Handshake Process - VALID and READY Signals: The source generates the VALID signal, indicating the availability of valid data. The destination generates the READY signal, indicating its readiness to accept the data. The actual transfer occurs when both the VALID and READY signals are high.

(5) Write Strobe: The WSTRB [n:0] signal indicates the valid bytes in each transfer, where the value of N depends on the size of each transfer. Read Response: This signal indicates the status of the read response.

(6) R/W Last: For RLAST, the slave asserts this signal only when driving the last transfer in the transaction. For WLAST, the master asserts this signal only when driving the last transfer in a transaction.

(7) B Channel: Write Response: This signal indicates the status of the write transaction.

Table 1 is the list of the signals used in the bridge design.

AW channel	**W channel**	**B channel**	**AR channel**	**R channel**
AW ADDRESS	W DATA	B RESP	AR ADDRESS	R DATA
AW LEN	W STRB	B READY	AR LEN	R RESP
AW SIZE	W LAST	B VALID	AR SIZE	R LAST
AW BURST	W VALID		AR BURST	R VALID
AW CACHE	W READY		AR CACHE	R READY
AW VALID			AR VALID	
AW READY			AR READY	

Table 1. AXI Signal List used in this work.

AXI-LIte protocol: Compared to the AXI protocol, the AXI-Lite protocol is more suitable for connecting to low-throughput components or peripherals such as UART. While AXI-Lite still utilizes five channels similar to AXI, some key differences exist. In AXI-Lite, the burst length is fixed at 1, meaning that each transfer in a transaction is always the last one. As a result, the RLAST and WLAST signals, which indicate the last transfer in a transaction in AXI, are not used in AXI-Lite. The size of each transfer in AXI-Lite can be either 32-bit or 64-bit, depending on the specific design requirements. This allows flexibility in handling data transfers based on the needs of the connected components or peripherals.

III. Bridge Implementation

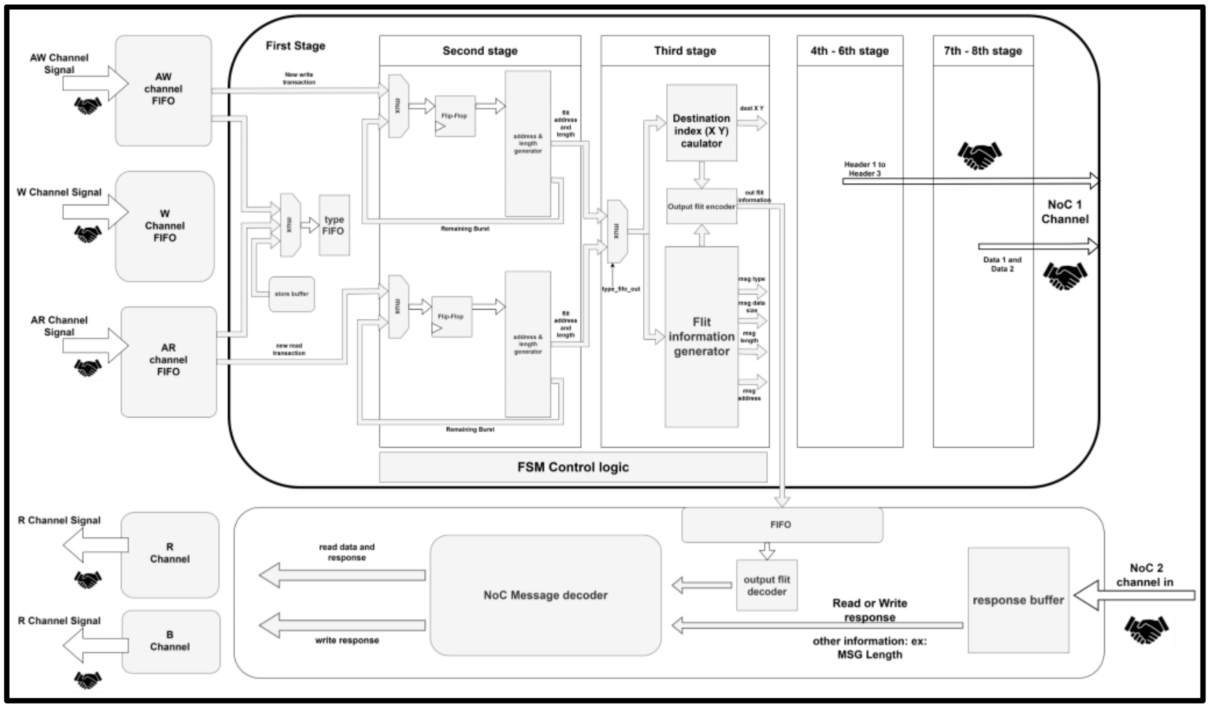


Figure 2. Bridge architecture

The AXI to NoC protocol bridge connects ARA's memory interface and the OpenPiton P-Mesh system, consisting of three cache levels: Ariane's private L1 cache, L1.5 cache, and distributed L2 cache. This bridge offers several features:

(1) Three different versions of the bridge are presented. The first version supports a 64-bit address width and either a 64-bit or 128-bit data width using the AXI protocol. The second version supports a 64-bit address width and uses the AXI-Lite protocol with a 64-bit data width.

(2) To fit ARA's requirements, the bridge only supports the INCREMENT burst type. This means that each transfer's address is an increment of the previous transfer with the size of the transaction.

(3) The bridge can handle up to a burst length of 256, allowing for efficient data transfers.

(4) The bridge supports up to 16 outstanding transactions simultaneously for both read and write channels, enabling parallel processing and improved performance.

On the NoC side, the output format of the bridge follows a specific structure, as illustrated in Figure 3. The width of the NoC channel is 64 bits, accommodating both header and data flits. Three header flits are required for a load request, containing the destination tile index, source tile index, address, data size, and message type. In the case of store requests, additional data flits are included along with the header flits. This configuration ensures efficient and reliable communication between the bridge and the NoC system.

In the OpenPiton P-Mesh system, two new message types, namely the Noshare-Load and Swap-Writeback operations, have been introduced in this work. These features were implemented for the first time and provided additional functionality to the system [5].

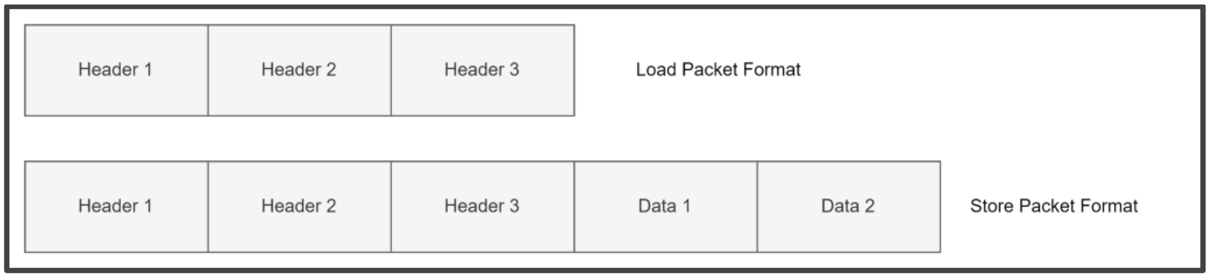


Figure 3. Load and Store packet format

(1) Noshare-load operation is the message type for loading requests sent to the L2 cache. When a Noshare-Load request is initiated, the L2 cache responds by providing a coherent copy of the requested data. The behavior of the L2 cache depends on the implementation of the directory-based MESI protocol. Several cases can be

considered: If the requested cache line in the L2 cache is in the E (Exclusive) or S (Shared) state, indicating that the cache line is owned or shared by other caches, the L2 cache sends a load request (Forward Read) to the owner or sharer. This step is necessary to handle the possibility of a silent eviction, where the owner or sharer may change the state of the cache line from E or S to M (Modified). Once the owner or sharer responds, the L2 cache returns the requested data to the requester.

(2) If the status of the requested cache line in the L2 cache is in the M (Modified) state, indicating that the cache line is exclusively modified in the L2 cache, the L2 cache downgrades the cache line to S state and returns the requested data.

(3) If the cache line is in the I (Invalid) state, indicating that the cache line is not present in the L2 cache, the L2 cache checks if it contains the requested cache line. If it does, the L2 cache directly returns the data. If not, the L2 cache loads and returns the data from the main memory.

(4) Swap-Writeback operation is the message type for store requests sent to the L2 cache. This operation is implemented similarly to the atomic operation in OpenPiton, ensuring system synchronization. However, the Swap-writeback operation provides higher granularity and includes a byte mask, reducing the throughput requirements and complexity of the bridge design. In addition to storing data at the specified address, the returned packet from the L2 cache also contains the previous value of the address. This feature is helpful for debugging purposes, verifying whether the value was correctly written into the specified address during the AXI burst transactions.

These new message types enhance the functionality and performance of the OpenPiton P-Mesh system, providing efficient and synchronized data loading and storing operations.

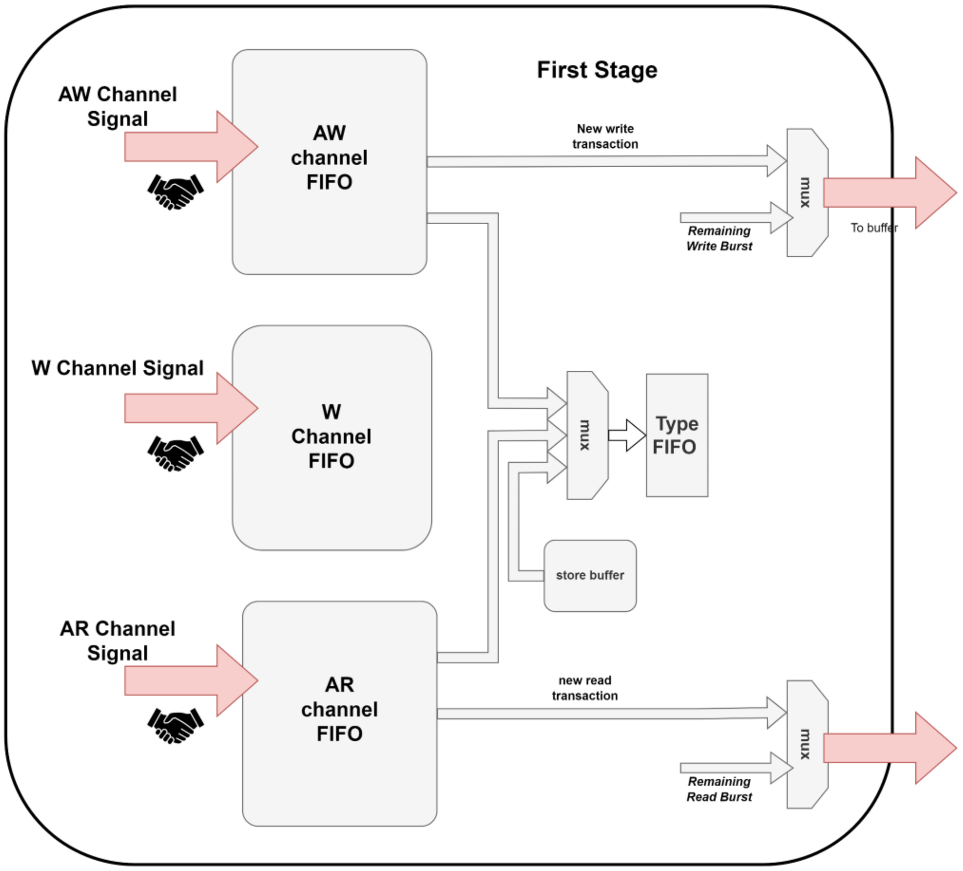


Figure 4. First stage

There are 8 stages in the bridge to packet the AXI request to OpenPiton NoC format. The first stage incorporates buffering mechanisms to handle concurrent read-write operations and ensure proper order of load and store requests per the AXI protocol. Figure 4 depicts multiple FIFOs (First-In-First-Out) employed in each channel to buffer the address, data, and control signals. To manage concurrent read-write operations, a store buffer mechanism is implemented. This mechanism prioritizes load requests over store requests since load requests typically require fewer cycles. The load-store order is maintained by processing load requests first and then addressing store requests. The "type fifo" buffer organizes the load and store requests sequentially, ensuring that the corresponding request is read from this FIFO when the transaction is sent out to the NoC interconnect. Lesson: A register slice could be inserted at any point of a different channel, with any possible additional cycles. There should be no fixed relationship (Like W ready depends on AW ready) between different channels. With those unnecessary dependencies, deadlock may happen due to the limitation of the buffer of each channel.

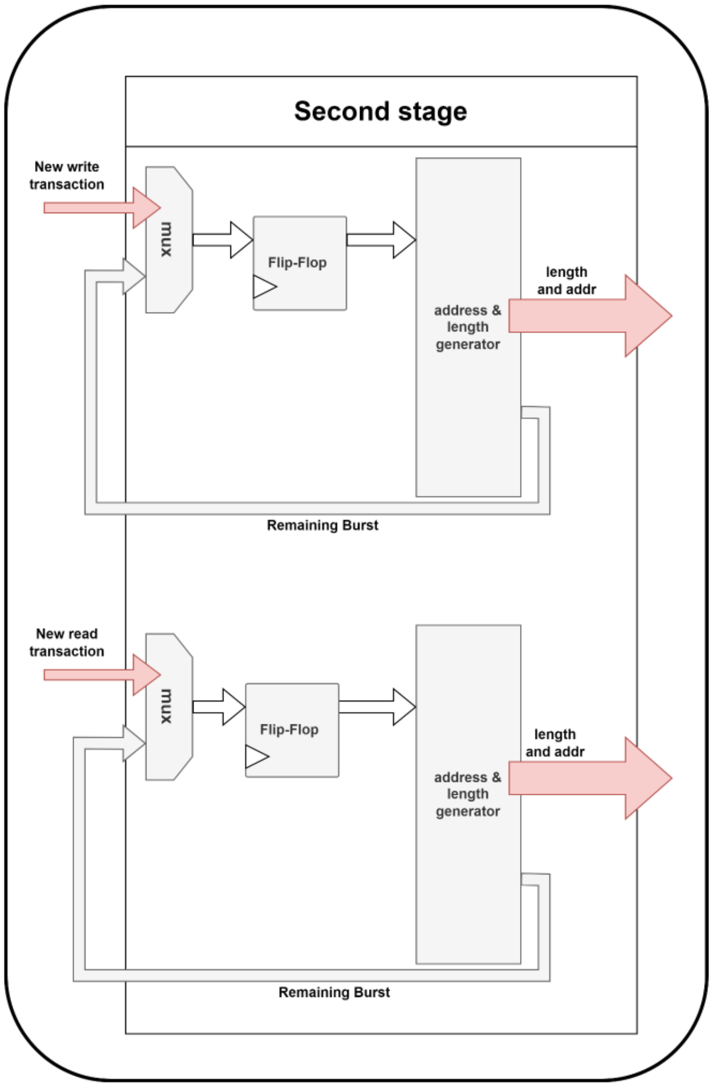


Figure 5. Second stage

The second stage in the design is responsible for determining the address and valid data length. In addition, the control logic within this stage adjusts based on the width of the bridge to comply with the 16-byte alignment restriction on the NoC packet side. When the data width is 8 bytes, the address is evaluated first to determine the location of the corresponding data within the NoC packet. If the address is aligned with 16 bytes, the data is placed in the first data flit. However, if the address is not aligned with 16 bytes, the data is placed in the second data flit. This information is also propagated to the read channel since the requested data is also 16 bytes in length. In the case of a 64-bit data width, if the burst length exceeds 2, the AXI transaction may need to be split into multiple NoC requests to accommodate the larger data size. Considering the data width, address alignment, and valid data length, the control logic within this stage ensures that the data is correctly placed within the NoC packet and that the AXI transaction is appropriately translated to meet the requirements of the NoC interconnect.

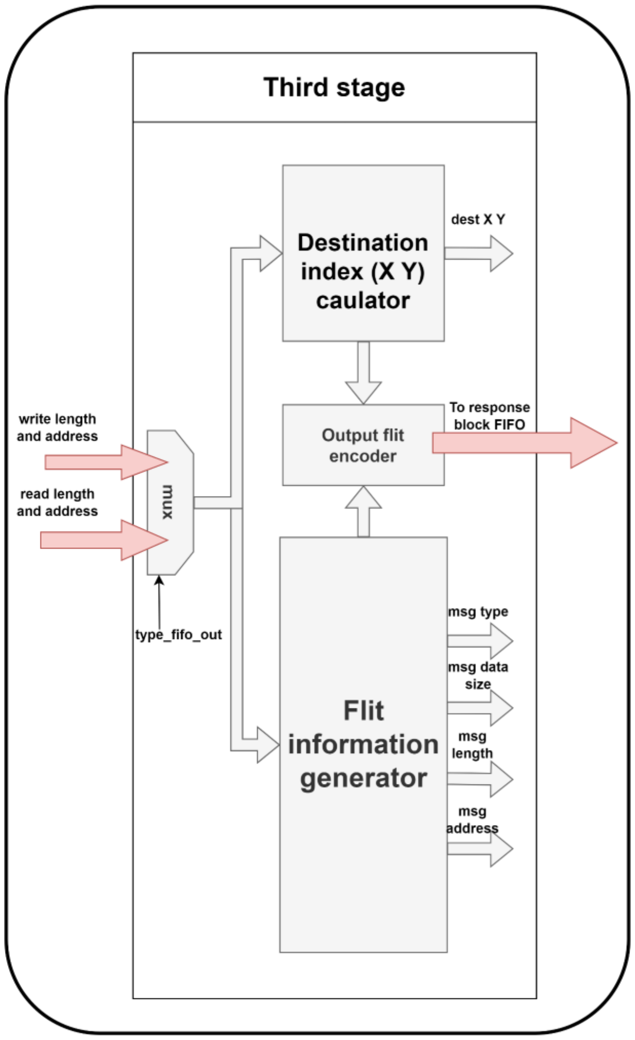
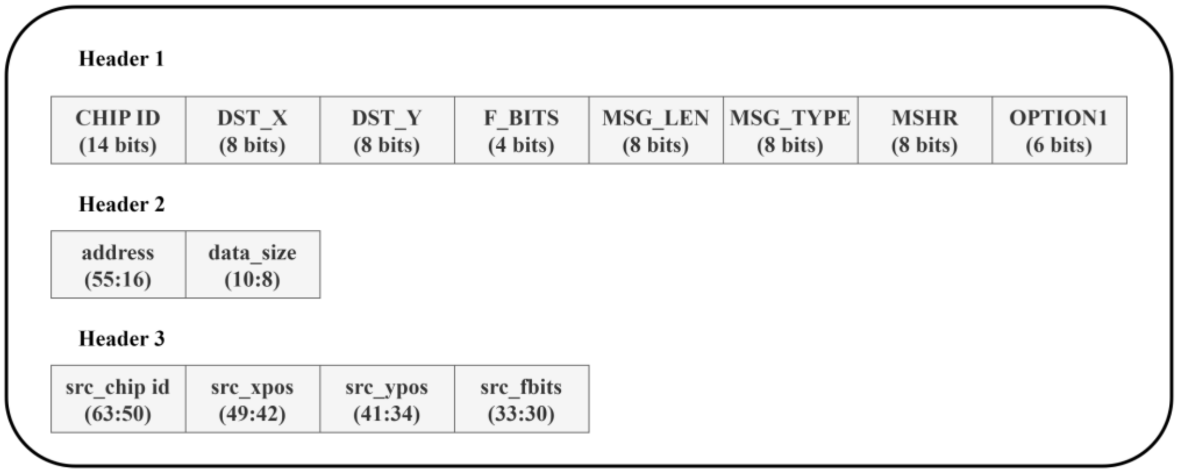


Figure 6. Third stage

The third stage of the design is responsible for resolving the destination tile index based on the tile number and address. Different addresses may correspond to different tiles' L2 cache, and this stage determines the appropriate destination tile index for the current request. Additionally, the flit information generator in this stage generates the necessary information required for the subsequent stages. Moving on to the fourth, fifth, and sixth stages, they aim to send out the three headers for store and load requests. These headers contain essential information that is used for communication within the system. The header format is illustrated in Figure 7 provided.

In these stages, the headers are constructed and transmitted. The destination tile indices (DEST_X and DST_Y) are obtained from the previous stage, which calculates the destination index. It is worth noting that for store requests, Header 2 and 3 contain additional fields for the bytes mask, allowing for more detailed information about which bytes are being stored.



Header 1

CHIP ID (14 bits)	DST_X (8 bits)	DST_Y (8 bits)	F_BITS (4 bits)	MSG_LEN (8 bits)	MSG_TYPE (8 bits)	MSHR (8 bits)	OPTION1 (6 bits)

Header 2

address (55:16)	data_size (10:8)

Header 3

src_chip id (63:50)	src_xpos (49:42)	src_ypos (41:34)	src_fbits (33:30)

Figure 7. Header flit format

The seventh and eighth stages of the design are responsible for handling the data flits in the communication process. Each data flit has a size of 64 bits.

In the seventh stage, the architecture contains the response block, as shown in Figure 7. This block receives the data flits, decodes them, and generates the correct data that needs to be sent to the corresponding AXI channel.

For load requests, the response block uses the information provided by the output flit decoder (from earlier stages) to pick the correct value from the received data flits. It ensures that the appropriate data is forwarded to the corresponding AXI channel for further processing.

In the case of store requests, the response block is responsible for generating the appropriate signals on the AXI B channel. This occurs once it receives the last response from the last transfer of the write transaction. The response block manages the synchronization and signaling required to indicate the completion of the store operation.

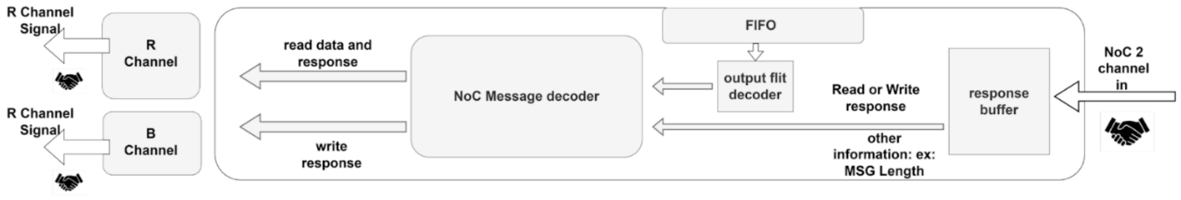


Figure 8. Response Block Architecture

The format of the return flit is shown in Figure 8. Note that for the store request, the data flit still returns data but is stale, which is the previous value of storing address.

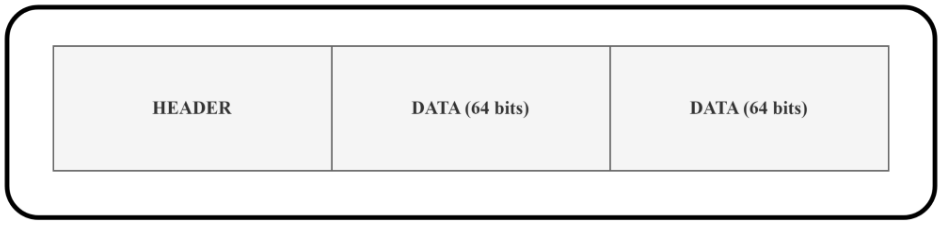


Figure 9. Response Flit Format

V. Result and performance evaluation

A. AXI-128 vs AXI-64

I conducted performance measurements on the system using two benchmarks: double-precision floating-point two-dimensional convolution (FCONV2D) and double-precision floating-point matrix multiplication (FMATMUL). The simulations were performed on Synopsys VCS 2020.03, and the test cases were compiled using LLVM 13.0.0. To accommodate the OpenPiton simulation environment, the original C and Python test cases were precompiled into binary files, enabling simulation with the "-precompiled" option.

The analysis focused on the memory interface and bridge design, with the ARA system having a fixed number of 4 total lanes and a vector register length of 4096. For the convolution computation, a fixed filter size of 3x3 was used. Convolution is a fundamental operation in convolutional neural networks, where a filter is applied to an input matrix to obtain a filtered output matrix. This operation involves the weighted sum of input elements. The performance evaluation of the convolution benchmark involved three different input matrix sizes with a fixed filter size of 3x3.

As depicted in Figures 10 and 11, the performance results indicate that the AXI-Lite bridge exhibits the highest computation cycles in both benchmarks, which can be attributed to the limited 64-bit data width on both the AXI and NoC sides, as well as the different data path of the AXI to AXI-Lite converter. However, the AXI-64 bridge demonstrates a significant improvement over the AXI-Lite bridge, and this advantage becomes more pronounced as the matrix size increases.

The AXI-128 bridge, which does not require a data width converter, avoids multiple cycles wasted in loading and storing datapaths. Nevertheless, the performance difference between the AXI-64 and AXI-128 bridges is nuanced. For instance, the table indicates that the AXI-64 bridge performs better when the matrix size is 16. Similar observations can be made for the FMATMUL benchmark.

This situation can be attributed to three main factors. Firstly, the enforced data alignment may result in fetching unused data, which decreases the throughput. In the case of the AXI-128 bridge, every transaction must be aligned with a 16-byte boundary, causing the load or store data to contain unused data, mainly when the transaction size is smaller than 128 bits, as permitted by the AXI protocol. Consequently, and the advantage of the entire 128-bit data width is not effectively utilized.

Secondly, the burst mechanism helps mitigate the limitations of the AXI-64 bridge. For example, when the burst length of transactions exceeds 2, and the address is aligned with a 128-bit boundary, the throughput of the AXI-64 bridge matches that of the AXI-128 bridge in such scenarios.

The third reason relates to the buffer capacity of the NoC channel, which may need to be increased for long burst lengths. In the OpenPiton L2 design, the NoC1 channel is used by L1.5 to issue requests, and it provides eight header flit buffers and four data flit buffers, as most of the input messages from NoC1 do not contain data. However, in the bridge implementation, only header flits are sent for load requests, while store requests require three header flits and two data flits. This utilization saturates all the available buffers in the NoC1 channel when dealing with very long burst-length AXI transactions. In the case of the AXI-128 bridge, the saturation further deteriorates the throughput on the NoC1 channel, as each transaction contains two data flits.

Furthermore, the bridges share the NoC2 channel with the L1.5 cache and the off-chip memory controller. As the matrix size increases, cache misses become more frequent, resulting in a high traffic volume on the NoC2 channel between the off-chip memory and the L2 cache. Simultaneously, the L2 cache uses the NoC2 channel to send responses to the bridge. This traffic saturates the output bandwidth of the L2 cache, even though the AXI-128 bridge can send out more packets.

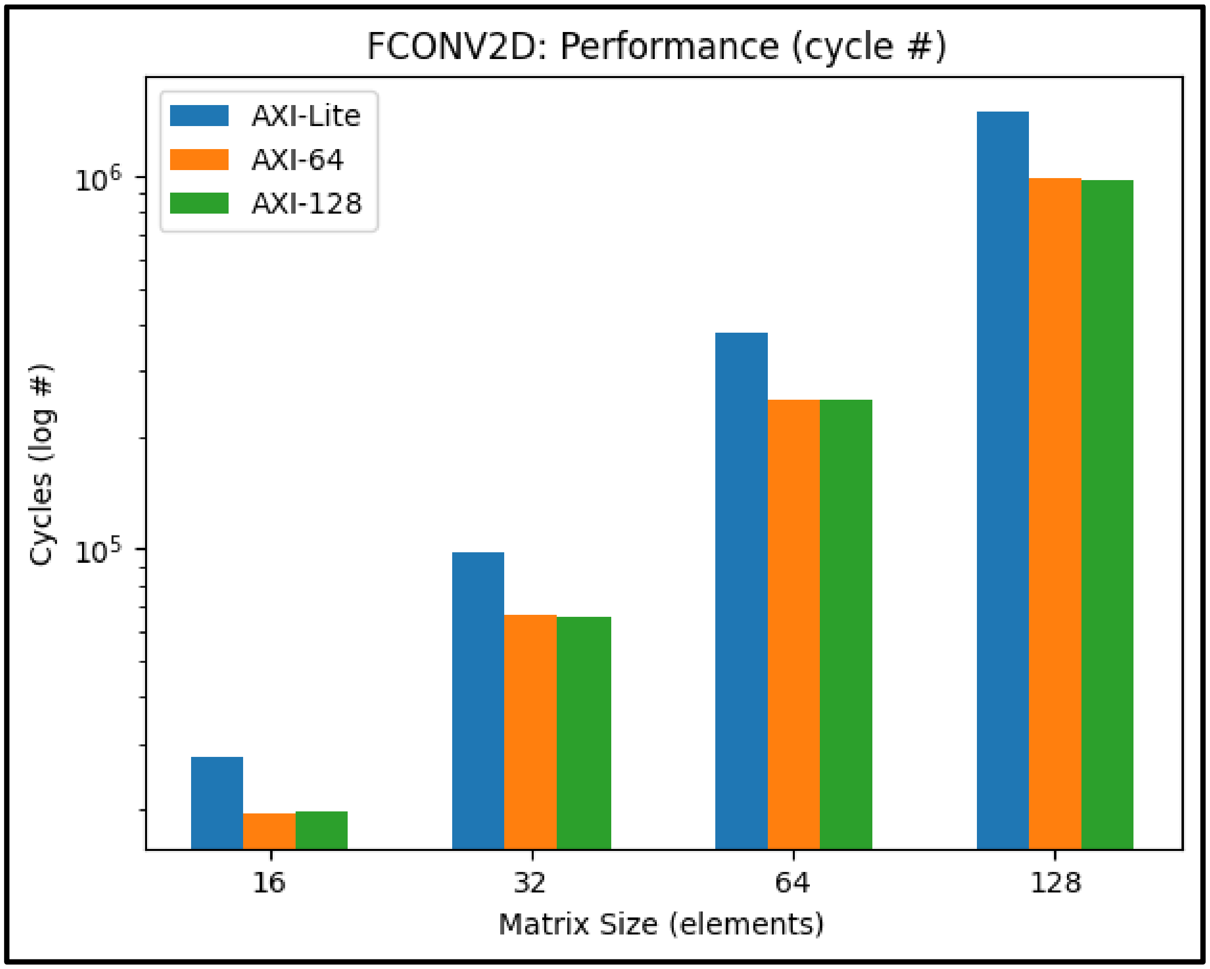


Figure 10. Three bridges comparison using FCONV2D benchmark, with different size of matrix.

FCONV2D: Performance Improvement		
Matrix Size (Elements)	AXI-64 vs AXI-Lite Speedup (%)	AXI-128 vs AXI-64 Speedup (%)
16	29.3	-1.09
32	32.2	0.93
64	33.5	0.52
128	34.3	0.37

Table 2. The cycle count improvement.

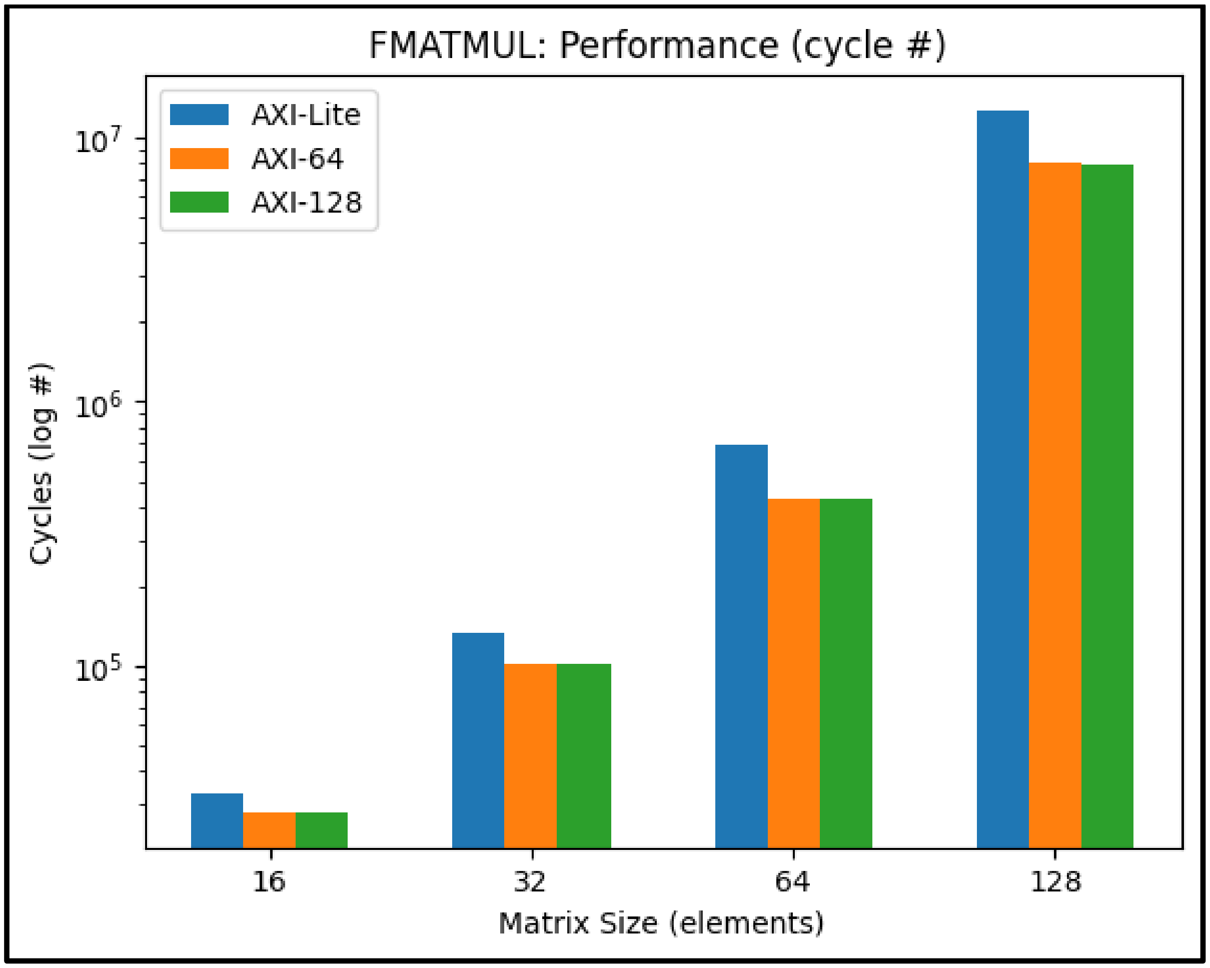


Figure 11. Three bridges comparison using FCONV2D benchmark, with different size of matrix.

FMATMUL: Performance Improvement		
Matrix Size (Elements)	**AXI-64 vs AXI-Lite speedup (%)**	**AXI-128 vs AXI-64 speedup (%)**
16 x 16	**14.4**	**0.19**
32 x 32	**24.0**	**0.30**
64 x 64	**36.7**	**0.032**
128 x 128	**37.1**	**0.48**

Table 3. The cycle count improvement.

B. Last Level Cache vs Off-Chip Memory

In addition to connecting to the L2 cache (Last Level Cache), I also evaluate the performance when Ara accesses the off-chip memory directly. In [8], the authors discussed three ways of connecting an accelerator to the CPU: tight-coupled accelerators (TCAs), loosely out-of-core coupling with Direct Memory Access to the Last-Level Cache (LLC-DMA), and loosely out-of-core coupling with Direct Memory Access to DRAM (DRAM-DMA). The integration of Ara is similar to the TCAs settings, which requires a special Instruction Set Architecture (ISA) to manage the accelerator's operation and may stall the CPU before the accelerator completes the instructions due to being tightly coupled with the CPU pipeline. However, in this work, Ara doesn't share L1 cache with Ariane; it connects to the LLC or off-chip memory. The load unit and store unit in Ara are also separated from Ariane. This configuration makes this work much similar to LLC-DMA and DRAM-DMA.

The comparison is based on two design models: AXI to NoC bridge with a 64-bit data width to the LLC (AXI64-LLC) and AXI to NoC bridge with a 64-bit data width to the off-chip memory (AXI64-MEM). FMATMUL and FCONV2D with different sizes are used for evaluation. The performance results in Figure 12 show the speedup ratio of AXI64-LLC over AXI64-MEM. For AXI64-MEM, it requires additional software cache flush to maintain function correctness, which definitely increases the total execution time. On the other hand, the results in [8] suggest that memory saturation occurs when running a large dataset with the coexistence of LLC-DMA and DRAM-DMA accelerators in one system. The LLC-DMA setting benefits from the mediation of LLC, which can reduce the risk of DRAM saturation. Similarly, in all the sizes of my evaluation, AXI64-LLC always takes fewer cycles than AXI64-MEM when executing both FMATMUL and FCONV2D benchmarks. The speedup of FCONV2D slightly decreases with the increasing matrix size. The arithmetic intensity of FONCV2D is not proportional to the matrix size but related to the size of the filter, hence the speedup for all sizes does not have a big variation.

The performance result of FMATMUL almost fits the roofline model [13], except for the performance and speedup ratio drop between 64 x 64 and 128 x 128. The arithmetic intensity of FMATMUL grows with O(n) as the matrix size increases, as shown in Figure 13. AXI64-MEM and AXI64-LLC show better performance, measured in DP-FLOP per cycle, with the increasing size of the matrix. The speedup ratio also widens the gap due to the low latency of L2 cache compared to the latency of off-chip memory. However, the ratio drops significantly for the 128 x 128 matrix size.

The default size of the LLC (L2 cache) in OpenPiton is 64KB, while for the 128 x 128 matrix with double precision floating-point elements, the total data size is 128KB. The large data size causes capacity misses during the execution of the FMATMUL benchmark. Additionally, the advantage of a large memory size becomes evident in this situation as it helps to hide the long latency compared to the smaller size benchmark. However, the decrease in the speedup ratio is attributed to the increasing miss rate in the LLC and the large volume of off-chip memory.

Moreover, when executing a 128x128 matrix, both benchmarks experience a significant decrease in performance. Unlike the findings in [8], the AXI64-LLC suffers from both the cache miss penalty and the additional latency of requesting off-chip memory, leading to memory saturation. Similarly, the performance of AXI64-MEM is also affected by memory saturation.

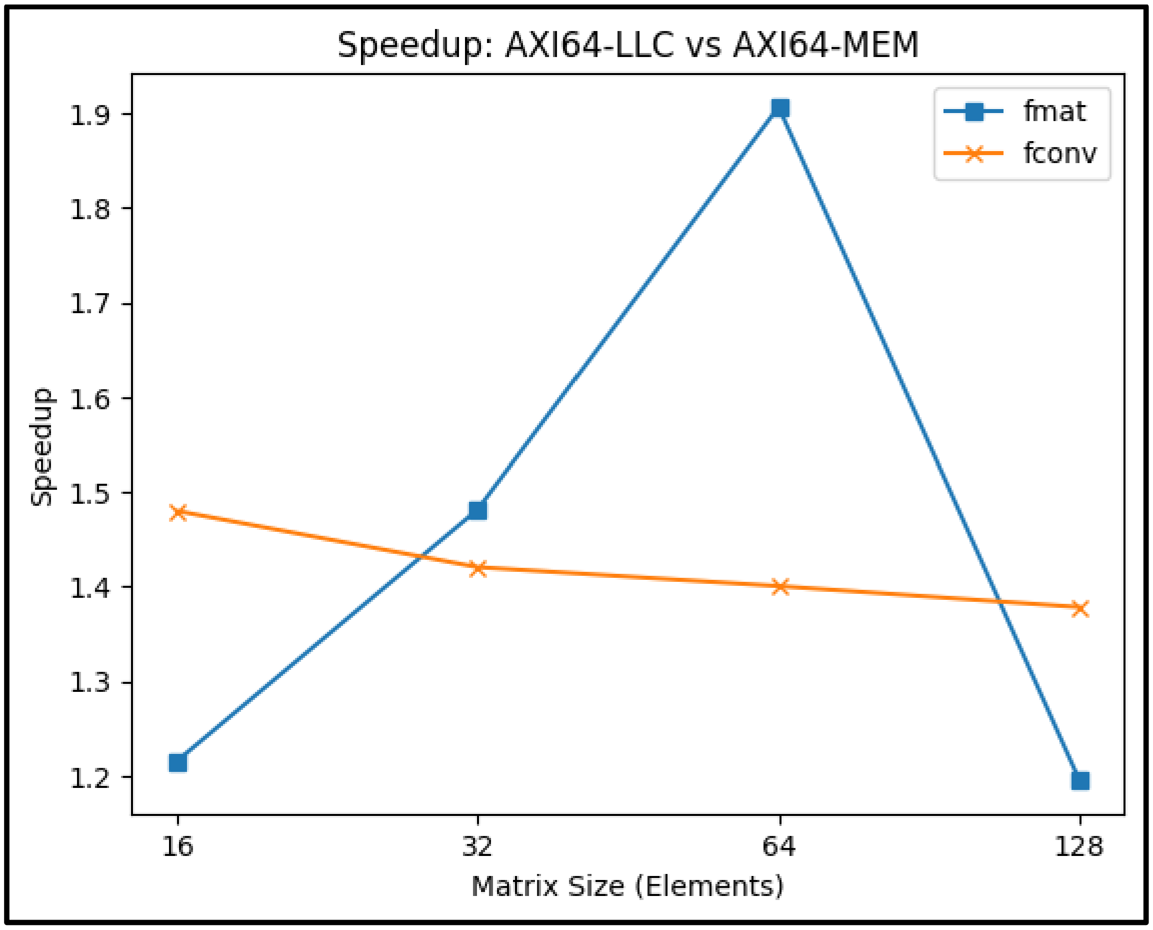


Figure 12. The speedup ratio: AXI64-LLC versus AXI64-MEM

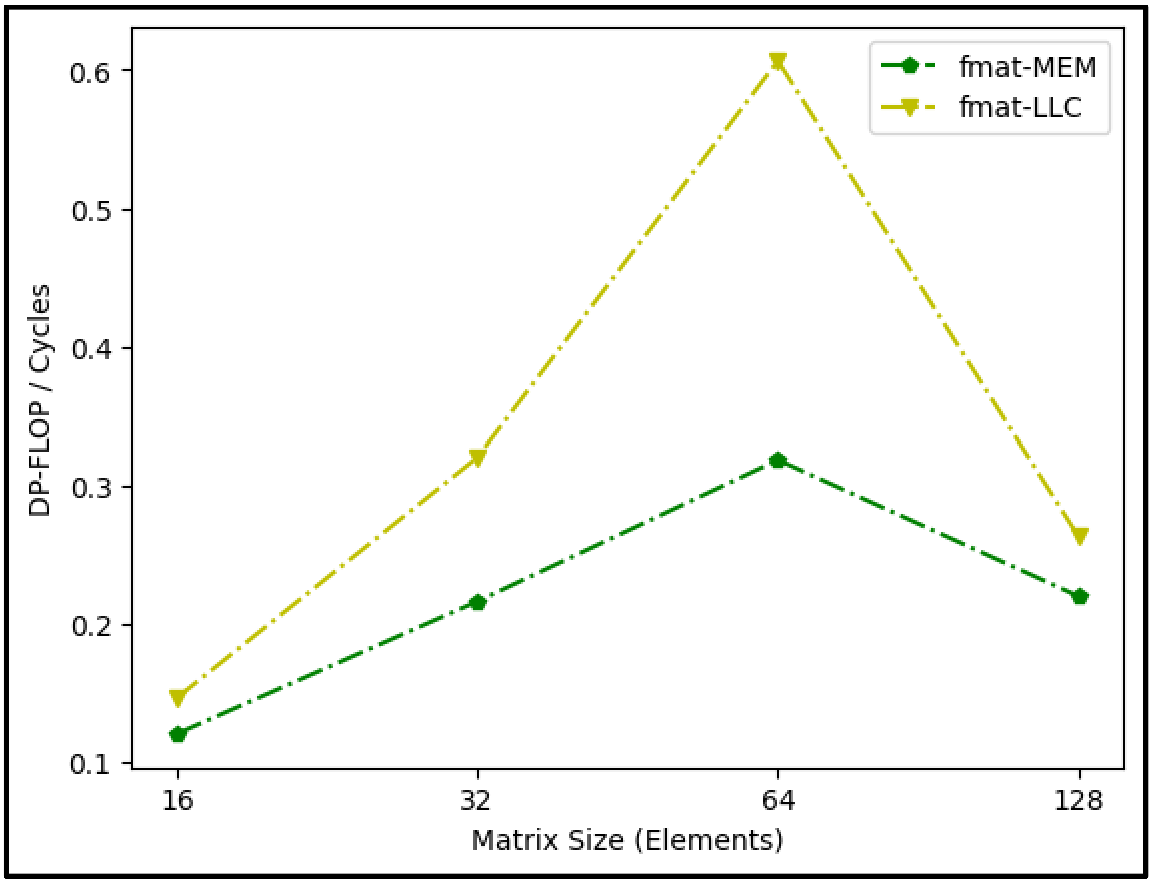


Figure 13. Performance result of double precision floating point matrix multiplication, with size differs from 16 x16 to 128 x 128.

VI. Future Work

One possible future direction is to enhance the throughput and outstanding transactions by incorporating Miss Status Holding Registers (MSHR) tags with the flits. This improvement is particularly beneficial when the transaction burst data size exceeds 64 bytes, which is the block size of the L2 cache. When requested data spans across cache blocks, the L2 cache may not maintain the acknowledgement order for both load and store operations due to cache misses, leading to disruption in the return order of available data. To address this issue, a structure similar to a reorder buffer, responsible for maintaining the instruction writing back orders, could be implemented in future designs.

Figure 14 illustrates the proposed structure based on the NoC Response block design. A reorder buffer is introduced for tag generation, incoming packet buffering, and dispatching the available response. The reorder buffer is designed as a queue structure with multiple blocks for storing incoming flit information, read data, and write acknowledgements. To ensure the order of return data or acknowledgements, an additional field called MSHR tag is utilized. Each sent-out packet is assigned an MSHR tag and stored in the reorder buffer. The corresponding response header flit contains the same MSHR tag, which is used in the reorder buffer to determine the availability of a specific response by matching the tags. Like the reorder buffer in CPU design, if the available response is at the head of the reorder buffer, it will be read out and sent to the AXI channel.

Furthermore, since tile structure is presented in this work, the extension to manycore, which is OpenPiton platform aiming for, can be implemented to increase the performance. In [6], they combined SIMD and MIMD through an instruction forwarding network mechanism, which mitigates the need for each core to fetch the instruction from its I cache. The vector processor is very efficient if the application's data stream is regular. This paradigm, MIMD, which manycore architecture represents, can increase performance unceasingly with the number of cores without heavy architecture changes at the same time. Though that the disadvantage of MIMD is exposed when facing high data parallelism applications. Each core tends to execute exact instructions with different data segments, resulting in wasted power and energy during instruction fetching [9]. Combining these SIMD and MIMD is worthwhile since real-world workloads are neither perfectly regular nor irregular.

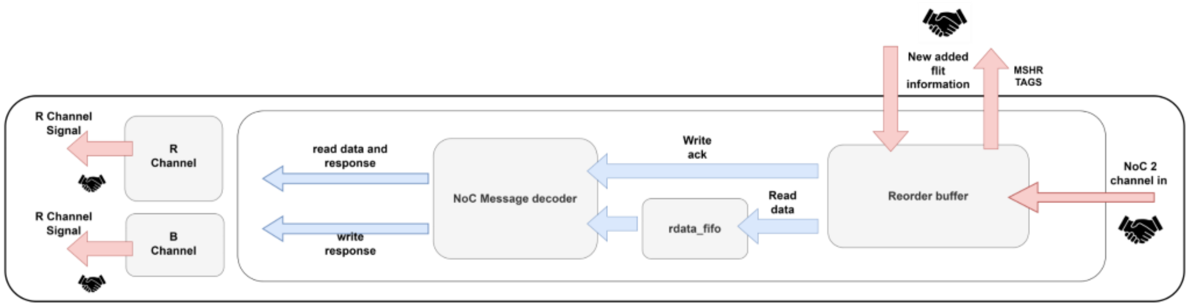


Figure 14. Structure of the new NoC response block for the future implementation.

VII. Conclusion

In this work, the focus is on designing AXI to NoC memory bridges for ARA memory access. These bridges provide the flexibility of future integration of other acclerators. Four versions of the bridges are presented: AXI-Lite bridge, AXI-LLC bridge with 64-bit and 128-bit data width (AXI-64 and AXI-128), and AXI-MEM bridge. These bridges support multiple AXI features such as outstanding transactions and bursts. They connect to the ARA AXI interface to receive memory requests and transform them into OpenPiton NoC packets directed to either the L2 cache or off-chip memory.

The performance evaluation is based on two benchmarks, FMATMUL and FCONV2D, with varying matrix sizes. For the bridges connecting to the LLC, the comparison primarily focuses on AXI-64 and AXI-128. The results show that the performance of these two bridges is almost identical. This outcome can be attributed to the unused data fetch, which causes bandwidth wastage in AXI-128, and the efficiency of burst operations in AXI, which enhances the performance of AXI-64 when executing benchmarks with large data volumes and consecutive address access.

Regarding the bridges connecting to off-chip memory, the results align closely with the roofline model presented in [13], with the exception of performance degradation observed in FMATMUL with a matrix size of 128x128. This degradation is caused by capacity misses in the L2 cache and memory saturation in the off-chip memory.

Though the framework this work is based on, OpenPiton, has connected NVDLA [2], an NVIDIA Deep Learning Accelerator, only the accelerator's register configuration is done by the NoC to AXI-Lite interface; NVDLA still accesses the data through its AXI interface, limiting efficiency and complexifying the hardware design. Hence, the flexibility of the AXI-to-NoC bridges presented in this work will help the future integration of accelerators, lower the design complexity, and increase the flexibility of the OpenPiton framework.

Finally, some future implementation direction is proposed, which involves the addition of a reorder buffer to achieve high throughput when dealing with multiple cache block accesses, and the extensibility to manycore architecture.

References

[1] A. AMBA and A. P. S. AXI. Axi4, and axi4-lite ace and ace-lite. ARM IHI D, 22.

[2] J. Balkind, T.-J. Chang, P. J. Jackson, G. Tziantzioulis, A. Li, F. Gao, A. Lavrov, G. Chirkov, J. Tu, M. Shahrad, et al. Openpiton at 5: A nexus for open and agile hardware design. IEEE Micro, 40(4):22–31, 2020.

[3] J. Balkind, K. Lim, M. Schaffner, F. Gao, G. Chirkov, A. Li, A. Lavrov, T. M. Nguyen, Y. Fu, F. Zaruba, et al. Byoc: a" bring your own core" framework for heterogeneous-isa research. In Proceedings of the Twenty-Fifth International Conference on Architectural Support for Programming Languages and Operating Systems, pages 699–714, 2020.

[4] J. Balkind, M. McKeown, Y. Fu, T. Nguyen, Y. Zhou, A. Lavrov, M. Shahrad, A. Fuchs, S. Payne, X. Liang, et al. Openpiton: An open source manycore research framework. ACM SIGPLAN Notices, 51(4):217–232, 2016.

[5] J. M. Balkind. Open Source Platforms for Enabling Full-Stack Hardware-Software Research. PhD thesis, Princeton University, 2022.

[6] P. Bedoukian, N. Adit, E. Peguero, and A. Sampson. Software-defined vector processing on manycore fabrics. In MICRO-54: 54th Annual IEEE/ACM International Symposium on Microarchitecture, pages 392–406, 2021.

[7] M. Cavalcante, F. Schuiki, F. Zaruba, M. Schaffner, and L. Benini. Ara: A 1-ghz+ scalable and energy-efficient risc-v vector processor with multiprecision floating-point support in 22-nm fd-soi. IEEE Transactions on Very Large Scale Integration (VLSI) Systems, 28(2):530–543, 2019.

[8] E. G. Cota, P. Mantovani, G. Di Guglielmo, and L. P. Carloni. An analysis of accelerator coupling in heterogeneous architectures. In Proceedings of the 52Nd Annual Design Automation Conference, pages 1–6, 2015.

[9] D. Dabbelt, C. Schmidt, E. Love, H. Mao, S. Karandikar, and K. Asanovic. Vector processors for energy efficient embedded systems. In Proceedings of the Third ACM International Workshop on Many-Core Embedded Systems, pages 10–16, 2016.

[10] J. A. Fisher. Very long instruction word architectures and the eli-512. In Proceedings of the 10th annual international symposium on Computer architecture, pages 140–150, 1983.

[11] J. L. Hennessy and D. A. Patterson. Computer architecture: a quantitative approach. Elsevier, 2011.

[12] A. Kurth, W. Rönninger, T. Benz, M. Cavalcante, F. Schuiki, F. Zaruba, and L. Benini. An open-source platform for high-performance non-coherent on-chip communication. IEEE Transactions on Computers, 71(8):1794–1809, 2021.

[13] M. Perotti, M. Cavalcante, N. Wistoff, R. Andri, L. Cavigelli, and L. Benini. A “new ara” for vector computing: An open source highly efficient risc-v v 1.0 vector processor design. In 2022 IEEE 33rd International Conference on Application-specific Systems, Architectures and Processors (ASAP), pages 43–51. IEEE, 2022.

[14] F. Zaruba and L. Benini. The cost of application-class processing: Energy and performance analysis of a linux-ready 1.7-ghz 64-bit risc-v core in 22-nm fdsoi technology. IEEE Transactions on Very Large Scale Integration (VLSI) Systems, 27(11):2629–2640, 2019.