FPGA-based Communication Protocols for High Performance Computing: Molecular Dynamics Simulation Application

M.S Research Thesis

By

ANKITKUMAR VIPULKUMAR PATEL

DEPARTMENT OF ELECTRICAL ENGINEERING

INDIAN INSTITUTE OF TECHNOLOGY INDORE

JUNE 2024

FPGA-based Communication Protocols for High Performance Computing: Molecular Dynamics Simulation Application

A Thesis

Submitted in partial fulfillment of the

requirements for the award of the degree

of

Master of Research

by

ANKITKUMAR VIPULKUMAR PATEL

DEPARTMENT OF ELECTRICAL ENGINEERING

INDIAN INSTITUTE OF TECHNOLOGY INDORE

JUNE 2024

INDIAN INSTITUTE OF TECHNOLOGY INDORE

CANDIDATE'S DECLARATION

I hereby certify that the work which is being presented in the thesis entitled **FPGA-based Communication Protocols for High-Performance Computing: Molecular Dynamics Simulation Application** in the partial fulfillment of the requirements for the award of the degree of **MASTER OF SCIENCE (RESEARCH)** and submitted in the **Department of Electrical Engineering, Indian Institute of Technology Indore,** is an authentic record of my own work carried out during the time period from August 2022 to June 2024 under the supervision of **Prof. Srivathsan Vasudevan**, Professor, Department of Electrical Engineering, Indian Institute of Technology, Indore and **Prof. Satya S. Bulusu**, Professor, Department of Chemistry, Indian Institute of Technology, Indore.

The matter presented in this thesis has not been submitted by me for the award of any other degree of this or any other institute.

Signature of the Student with Date

(Ankitkumar Vipulkumar Patel)

This is to certify that the above statement made by the candidate is correct to the best of my knowledge.

14/09/2024

Signature of Thesis Supervisor with Date

(Prof. Srivathsan Vasudevan)

14/09/2024

Signature of Thesis Supervisor with Date

(Prof. Satya S. Bulusu)

Ankitkumar Vipulkumar Patel has successfully given his MS (Research) Oral Examination held on _____

Signature of Chairperson, OEB with Date

Signature of Thesis Supervisors with date

Signature of Convener, DPGC with date

Signature of Head of Department with date

ACKNOWLEDGEMENTS

First and foremost, I would like to express my sincere appreciation to my thesis supervisors, **Prof. Srivathsan Vasudevan**, Professor, Department of Electrical Engineering, IIT Indore, and **Prof. Satya S. Bulusu**, Professor, Department of Chemistry, IIT Indore. Their generous guidance, support, mentoring, constant encouragement, and invaluable suggestions have been instrumental in making this work possible.

I am extremely grateful to the MS (Research) Coordinator, **Prof. Trapti Jain**, for her assistance from the admission process to the final evaluation. Her valuable advice, insightful comments, and cooperation throughout my research have been greatly appreciated.

I am deeply thankful to **IIT Indore** for providing essential lab facilities and supporting my academic endeavors.

I would also like to express my heartfelt gratitude to my parents, **Vipulkumar Keshavlal Patel** and **Mrs. Ashaben Vipulkumar Patel**. Their excellent guidance, unwavering support, and belief in me have been the pillars of my strength. I am also thankful to my sister, **Mrs. Jaimini Rohit Patel**, for her continuous encouragement and unconditional support.

I would like to extend my wholehearted thanks to **Mr. Abhishek Ojha** and **Ms. Aparna Gangwar** for their trustworthy help, discussions, unwavering support, and for creating a positive environment in the lab during challenging times. I also thank **Mr. Kishore Reddy Kurapati** and **Mr. Dharmendra Kartikey** for their assistance in my research work. My gratitude also extends to my other lab members, **Dr. Suhel Khan**, **Mr. Pracheta Chatterjee**, **Ms. Prerna**, and **Mr. Bhusan Borah**, for their support whenever needed.

Last but not the least, I am thankful to my friends, without whom this journey wouldn't have been as enjoyable. All the tea breaks, mall trips, dancing, and campus wanderings with **Shruti Ghodke**, **Komal Gupta**, **Sai Ganesh**, **Radheshyam Sharma**, **Neeraj Nikhil**, **Gulrej Khan**, **Anupma Sharma** and many others will be the most memorable part of my M.S (Research) life.

I am sincerely thankful to the Ministry of Human Resource Development (MHRD), Government of India, and the Council of Scientific and Industrial Research, Government of India, for providing me the fellowship to pursue this research work.

Ankitkumar Vipulkumar Patel

This Thesis is Dedicated

to

My Family and Friends

Abstract

Molecular Dynamics (MD) simulation for computing Interatomic Potential (IAP) is a very important High-Performance Computing (HPC) application. MD simulation on particles of experimental relevance takes huge computation time despite using an expensive high-end server. Heterogeneous computing, a combination of FPGA and a computer, is proposed as a solution to compute MD simulation efficiently. In such heterogeneous computation, communication between FPGA and Computer is necessary. One such MD simulation, explained in the paper, is the ANN-based IAP computation of gold (Au_{147} & Au_{309}) nanoparticles. MD simulation calculates the forces between atoms and the total energy of the chemical system. This work proposes the novel design and implementation of an ANN IAP-based MD simulation for Au_{147} & Au_{309} using communication protocols, such as UART and Ethernet, for communication between the FPGA and the host computer. To improve the latency of MD simulation through heterogeneous computing, UART and Ethernet communication protocols were explored to conduct MD simulation of 50,000 cycles. In this study, computation times of 17.54 hours and 18.70 hours were achieved with UART and Ethernet, respectively, compared to the conventional server time of 29 hours for the Au_{147} atomic cluster. The results pave the way for the development of a Lab-on-a-chip application.

LIST OF PUBLICATIONS

- **Journal Publications:**

1. **Ankitkumar Patel**, Srivathsan Vasudevan, & Satya S. Bulusu. "FPGA Accelerators for Computing Interatomic Potential-based Molecular Dynamics Simulation for Gold Nanoparticles: Exploring Different Communication Protocols". Computers, Materials & Continua (CMC). (IF 3.1)

Contents

List of Figures

List of Tables

LIST OF ABBREVIATIONS

HPC	-	High Performance Computing
CPU	-	Central Processing Unit
GPU	-	Graphics Processing Unit
ASIC	-	Application Specific Integrated Circuit
FPGA	-	Field Programmable Gate Array
MD	-	Molecular Dynamics
ANN	-	Artificial Neural Network
IP	-	Intellectual Property
CLB	-	Configurable Logic Block
LUT	-	Look Up Table
MUX	-	Multiplexer
HDL	-	Hardware Design Language
AXI	-	Advanced eXtensible Interface
HLS	-	High Level Synthesis
RISC	-	Reduced Instruction Set Computer
IAP	-	Interatomic Potential
DSP	-	Digital Signal Processing
PCIe	-	Peripheral Component Interconnect Express

UART	-	Universal Asynchronous Receiver-Transmitter
SRAM	-	Static Random Access Memory
IO	-	Input Output
BRAM	-	Block Random Access Memory
SDK	-	Software Development Kit
DDR	-	Double Data Rate
SPI	-	Serial Peripheral Interface
OSI	-	Open System Interconnect
LAN	-	Local Area Network
LLC	-	Logical Link Control
MAC	-	Medium Access Control
NIC	-	Network Interface Card
CRC	-	Cyclic Redundancy Check
IC	-	Instruction Cache
DC	-	Data Cache
FIFO	-	First In First Out
MII	-	Media Independent Interface
MIG	-	Memory Interface Generator
RTL	-	Register Transfer Level

Chapter 1
Introduction

1.1 Field Programmable Gate Array (FPGA)

Field-programmable gate arrays (FPGAs) are reconfigurable computer chips that can be programmed to implement any digital hardware circuit. FPGAs consist of an array of programmable blocks (logic, I/O, and others) that can be interconnected using prefabricated routing tracks with programmable switches. The functionality of all FPGA blocks and the configuration of the routing switches are controlled by millions of static random-access memory (SRAM) cells, which are programmed at runtime to realize a specific function. The user describes the desired functionality in a hardware description language (HDL) such as Verilog or VHDL, or possibly uses high-level synthesis to translate C or OpenCL to HDL. The HDL design is then compiled using a complex computer-aided design (CAD) flow into a bitstream file used to program the FPGA's configuration SRAM cells [1].

Compared to developing a custom application-specific integrated circuit (ASIC), FPGAs offer significantly lower initial engineering costs and quicker time-to-market. Using a pre-made off-the-shelf FPGA, a complete system can be set up within weeks, bypassing the need for physical design, layout, fabrication, and verification typical of custom ASICs. Moreover, FPGAs allow for seamless hardware updates post-deployment by simply loading a new bitstream in the field, hence the name "field-programmable." This feature makes FPGAs an attractive option for medium and small-scale projects, especially given the rapid product turnover in today's markets. These benefits have encouraged the use of FPGAs in various fields, such as wireless communications [2], embedded signal processing [3], networking [4], ASIC prototyping [5], and high-frequency trading [6].

Due to its parallel computing feature, FPGA is the most attractive choice for the heterogeneous computing platform.

1.2 High Performance Computing (HPC)

High performance computing (HPC) is a way to process large amounts of data that goes through some computing algorithm at high speeds to solve complex problems in engineering, science, or business fields in a broad context. HPC systems can perform calculations much faster than traditional computers and can be more powerful than a single workstation, server, or computer [7]. To achieve HPC computation, processors/clusters work simultaneously to handle large-scale multi-dimensional datasets to solve complex problems at very high speeds [8].

HPC performance is typically measured using two fundamental metrics: "time" and "number of operations." The most commonly used metric in HPC is "floating point operations per second," or "flops" [9]. While advancements in semiconductor technology continue to push the boundaries of computing, this progress also brings manufacturing closer to physical limits. Managing heat and power becomes increasingly challenging as processor speeds and core counts increase. In response to evolving user needs and power constraints, heterogeneous computing platforms have emerged as a viable solution [10].

1.2.1 Heterogeneous Computing

With the tremendous increase in data volume, traditional CPUs are finding it challenging to keep up with the demands of HPC [11]. Researchers are working on improving the speed of HPC applications. With advancements in the VLSI hardware industry, they use heterogeneous computing platforms to improve the performance of these HPC applications [12]. Heterogeneous computing is a unique method of handling many tasks simultaneously by spreading them across different systems. This not only boosts performance but also saves power [10] [13]. It moves beyond Moore's Law and tackles issues like power usage, per-

formance, and scalability. A heterogeneous platform efficiently manages tasks using different computing units, finding the right balance between performance and power. These platforms employ various computing units like CPUs, GPUs, DSPs, ASICs, and FPGAs [14] [15] [16]. Among these, FPGAs stand out for their ability to handle multiple tasks simultaneously, speeding up computations [17]. Figure 1.1 shows the GPU-CPU and FPGA-CPU heterogeneous computing platforms. In contrast, homogeneous computing platforms use the same type of computing units for every task.

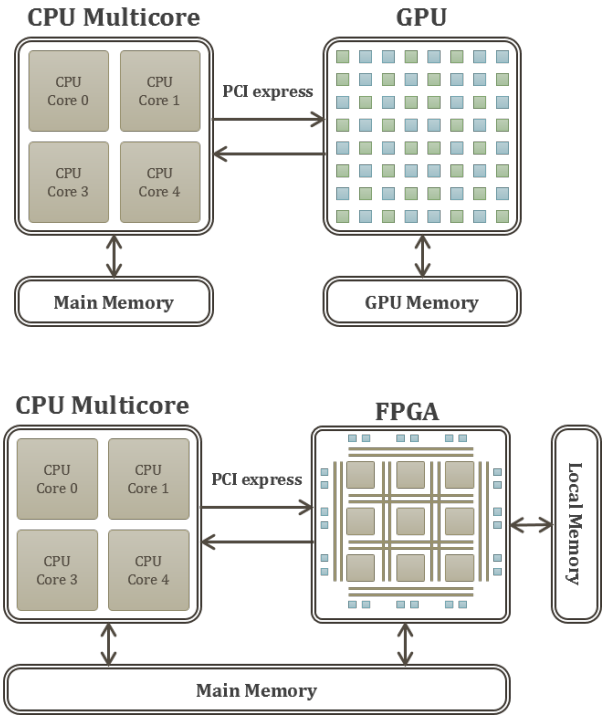


Figure 1.1: GPU-CPU and FPGA-CPU Heterogeneous Computing Platform

Although CPUs are great for handling tasks sequentially and FPGAs for parallel tasks, they are essential in heterogeneous computing setups. However, FPGA-CPU heterogeneous computing platforms have limitations, with a significant drawback being the communication overhead between FPGA and CPU.

Initially, our group developed an FPGA accelerator for machine learning IAP-based molecular dynamics of gold (Au_{13}, Au_{55}, Au_{147}, Au_{309}, etc.) nanoparticles using the phenomenon of heterogeneous computing [18]. The block diagram of hardware-software co-design is shown in Figure 1.2. For implementation, they utilized the PCIe communication protocol for communication between FPGA and CPU. Gold nanoparticles were chosen because they have always been a subject of interest in various applications like biomedical, chemical, plasmonics, and non-linear optics, among others [19].

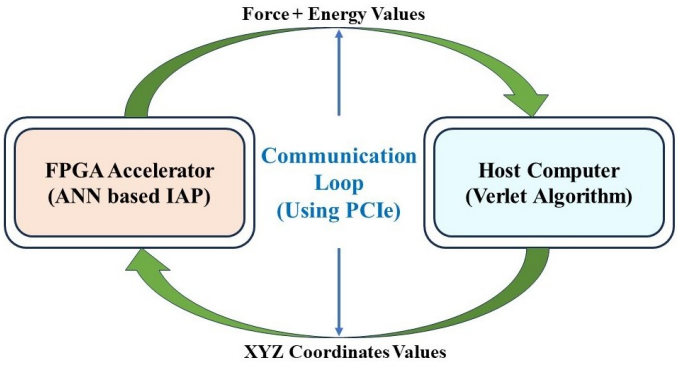


Figure 1.2: FPGA Accelerator Hardware-Software Co-design Using PCIe Communication

Molecular dynamics is a routinely used High Performance Computing(HPC) technique, which involves many computers to perform the task of obtaining the dynamic properties of atoms and molecules. For such calculations, the bottleneck usually involves computing forces (energy derivatives) for each particle at every time step. To get equilibrium properties, we have to run the simulations for more than 3 million steps.

Initially, this ML-IAP based MD ran on an HPC Server. But, In HPC servers, several processors run in parallel, consuming a large amount of power and incurring high capital and maintenance costs. To address these issues, researchers are exploring the concept of FPGA-CPU heterogeneous computing platform[20].

1.3 FPGA Accelerator: Using PCIe Communication Protocol

In hardware accelerator architecture, the ML-IAP based MD program is divided into two parts: a) calculations of IAP-based Intellectual Property (IP) block for energy and atomic forces that will be implemented on FPGA and b) integration of Newton's equations of motion that will be executed on a computer (CPU) [18]. Besides, there exist several dependencies in the algorithm which restrict concurrent computation. Identifying each and every step of such processes and paving the way for efficient parallelization involves detailed analysis of timing diagrams and proposing alternative programming options without compromising precision and resource utilization.

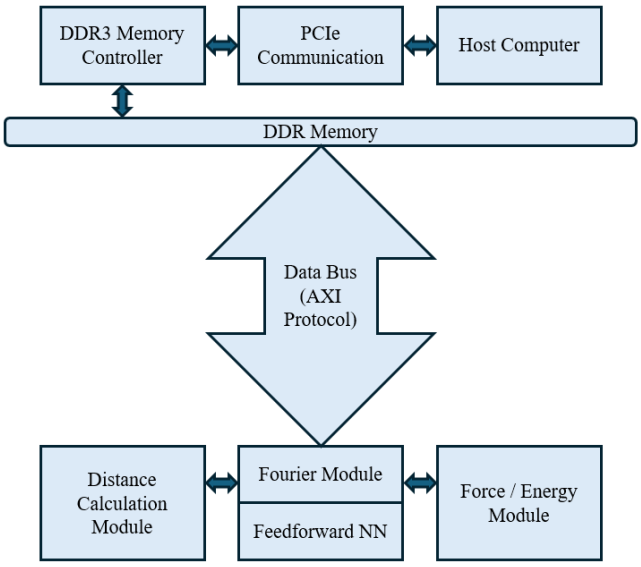


Figure 1.3: System Architecture of FPGA Accelerator using PCIe Communication [18]

The system architecture of the hardware-software co-design, shown in Fig-

ure 1.3 [18], divides the system into two parts. The FPGA hardware handles complex, time-intensive computations while the host computer manages control functions and sequential computations. The heterogeneous computing-based MD simulation model requires constant communication between the host computer and the FPGA board for proper operation. This communication is facilitated using the PCIe protocol. The DMA subsystem for PCIe allows the host computer to access memory residing on the FPGA board. During computations on the FPGA, certain tasks require large floating-point arrays, which are stored in DDR memory. The AXI protocol is used for communication between various IP cores on the FPGA chip.

In every MD cycle, the Cartesian coordinates (X, Y, and Z) are sent from the host computer to the FPGA, and the outputs (forces and energy values) are sent from the FPGA to the host computer using PCIe communication. Thus, around 3.5KB of data transfer occurs between the FPGA and a host computer in every MD cycle using PCIe for Au_{147}. With PCIe communication 50,000 MD cycles, PCIe took 19.84 hours, whereas HPC Server took 29 hours for Au_{147} atomic clusters. With PCIe, the computation time was reduced by 1.5 times, as reported by Bulusu et al. [18].

1.3.1 PCIe Communication Protocol

PCIe is a serial expansion bus standard used for connecting a computer to one or more peripheral devices. Compared to parallel buses like PCI and PCI eXtended, PCIe offers lower latency and higher data transfer rates. Each device connected to a motherboard via PCIe has its own dedicated point-to-point connection. This means that expansion card devices don't have to compete for bandwidth because they aren't sharing the same bus [21].

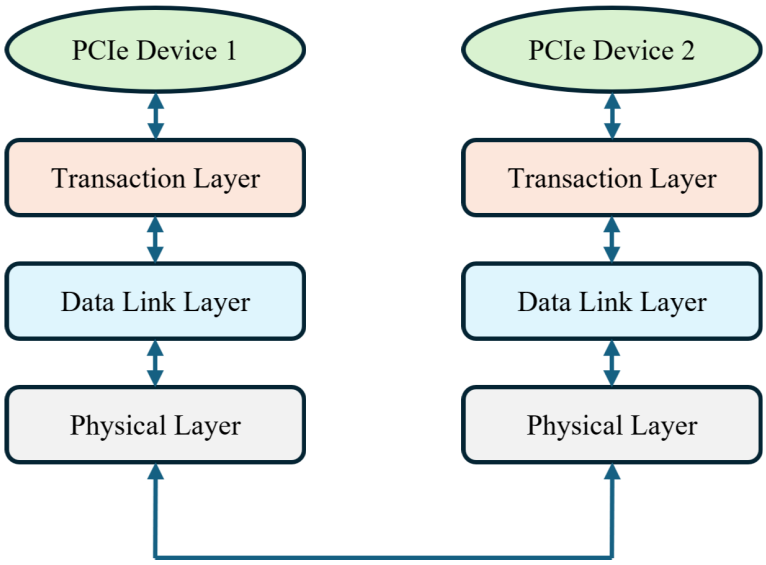


Figure 1.4: PCIe Protocol Stack.

PCIe achieves reliable data transfers using a three-layer PCIe protocol stack as shown in Figure 1.4.

1. Physical Layer specifies the electrical and logical design. It represents the size and shape of the PCIe cards, slots, pin configuration and the use of differential signal pairs.

2. Data Link Layer specifies the ways in which data is packaged, sequenced and moved. It handles the initialization, data movement and signaling between the PCIe device and the host interface.

3. Transaction Layer is the highest and most abstract layer of a PCIe standard that defines the content – how data is structured and what's contained within each package of PCIe data as they move between points.

PCIe is suitable for a wide range of applications across various sectors such as Communications, Datacenter, Enterprise, Embedded, Test & Measurement,

Military, and others. It serves as a peripheral device interconnect, a chip-to-chip interface, and a bridge to many other protocol standards.

1.3.2 Limitations of PCIe Communication Protocol

A thorough study by Kucharczyk et al. [13] on FPGA applications points out PCIe problems. First, it says we need special hardware and drivers, like a PCIe slot and its drivers, which make programming more complicated. Second, it mentions that the PCIe cannot support hot plug operations, requiring a system restart for reconfiguration. Also, PCIe for small data transfers, like 3.5KB for Au_{147} and 7.2KB for Au_{309}, adds unnecessary complexity. Hence, the advantage of high-speed and low-latency data transfer is reduced due to the complexity overhead in small KB data transfers.

1.4 Objectives

1. Explore UART and Ethernet communication protocols as alternatives to PCIe communication for ANN-based MD simulation for gold nanoparticles. The emphasis is on implementing ANN interatomic potential-based MD simulations, prioritizing simplicity, user-friendliness, performance, and energy consumption efficiency.

2. Integrate the Microblaze soft-core processor to enhance hardware control, ensuring a system that is easily debuggable and well-controlled.

3. Implement the MD calculation in the FPGA-Computer (CPU) based system using both communication protocols.

1.5 Organization of the Thesis

This section provides an overview of how the thesis is organized. The thesis consists of four chapters whose contents are as follows:

Chapter 1 introduces the various topics necessary to understand the need for the thesis work. It also describes the motivation and objectives of the thesis.

Chapter 2 explores the theoretical fundamentals and literature survey required to develop the FPGA Accelerator for IAP-based MD Simulation.

Chapter 3 details the hardware and software development and implementation on the Kintex-7 KC705 FPGA board.

Chapter 4 presents the results and discussion. It concludes with a summary and explores the future scope of the work.

Chapter 2
Literature Survey

2.1 FPGA Architecture and Design Flow

2.1.1 FPGA Architecture

Today's FPGAs feature specialized components tailored for specific tasks alongside flexible configurable logic. This mix of dedicated functionality and adaptable logic contributes to architectures that boast lower power consumption and enhanced performance [22] [23] [24]. The basic FPGA architecture is shown in Figure 2.1.

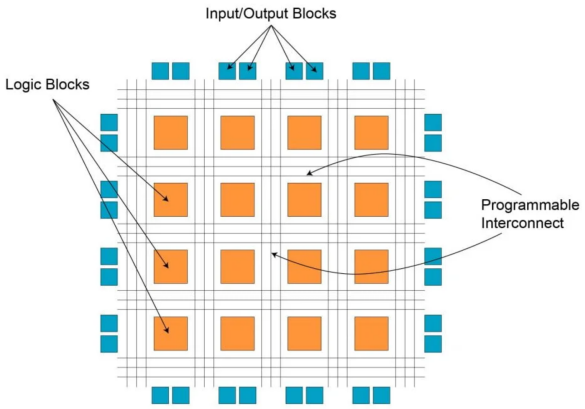


Figure 2.1: FPGA Architecture.

Configurable Logic Block (CLB)

A CLB is the core component of an FPGA, enabling it to adopt various hardware setups. Essentially, an FPGA is comprised of numerous CLBs, forming its structure. With thousands of these on modern FPGAs, they can be programmed to execute nearly any logical operation. Each CLB contains its own set of discrete logic elements, including look-up tables (LUTs), multiplexers, and flip-flops [25] [26]. The basic CLB structure is shown in Figure 2.2.

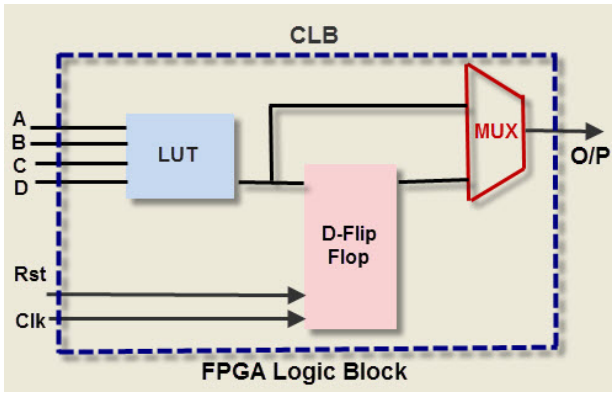


Figure 2.2: FPGA Configurable Logic Block.

1. **Look-up Table**

   The Lookup Table (LUT), a pivotal element within the CLB of an FPGA, serves as a cornerstone for its functionality. It operates as a customizable truth table, associating input configurations with corresponding output values. LUTs possess programmable capabilities, enabling them to execute various combinational logic functions. Their flexibility is paramount in digital circuit design, as they empower engineers to implement diverse logic operations tailored to specific requirements. From basic logic gates to intricate functions such as adders and multiplexers, LUTs provide a versatile platform for crafting custom digital circuits, accommodating a spectrum of computational tasks with precision and efficiency [25] [26].

2. **Flip-flop or Latches**

   Flip-flops or latches, integral components within a CLB, play a vital role in FPGA functionality by providing the capability to both store and synchronize data. These flip-flops or latches serve multiple purposes within the digital circuitry: they can store intermediate results during complex computations, implement sequential logic to control the order of operations and create memory elements for temporary or permanent data stor-

age. Their versatility enables the FPGA to manage and manipulate data streams effectively, enhancing its overall performance and functionality in various applications [25] [26].

3. **Multiplexer (MUX)**

   The multiplexer (MUX) housed within a CLB is critical in managing data flow within an FPGA. It acts as a switchboard, enabling the selection of different inputs based on control signals. This capability allows signals from various sources to be routed and connected to different components within the CLB as needed. By providing this flexibility, the multiplexer facilitates the adaptation of the FPGA to diverse operational requirements. For instance, it permits the selection of appropriate input signals for specific computational tasks or conditions, thereby enhancing the versatility and efficiency of the FPGA's operation. Additionally, the MUX is crucial in optimizing resource utilization within the CLB, ensuring that signals are efficiently directed to the appropriate destinations for processing or further routing [25].

Programmable Interconnects

In FPGAs, routing consists of wire segments of varying lengths interconnected by electrically programmable switches. The density of logic blocks within an FPGA is influenced by the length and number of these wire segments used for routing. Balancing the number of connecting segments is crucial as it affects both the density of logic blocks and the space occupied by routing. Programmable routing links logic and input/output blocks to complete a user-defined design unit. This routing circuit comprises multiplexers, pass transistors and tri-state buffers. Pass transistors and multiplexers within a logic cluster establish connections between the logic units [22] [23].

Programmable I/O Blocks

FPGAs are equipped with adaptable IO structures designed to facilitate communication with a diverse range of devices, establishing them as pivotal communication hubs in numerous systems. However, accommodating multiple IO interfaces and standards using a single set of physical IOs presents significant challenges, requiring adaptation to varying voltage levels, electrical characteristics, timing specifications, and command protocols [22] [23].

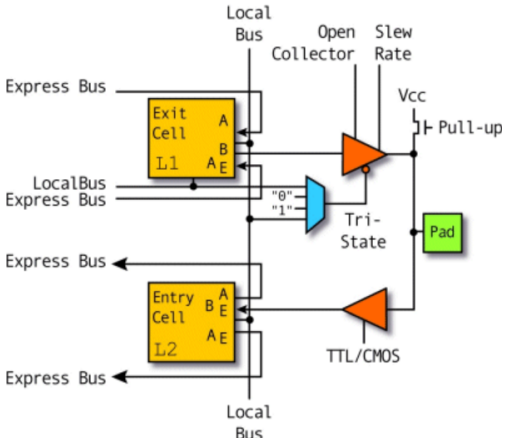


Figure 2.3: FPGA Programmable I/O Block.

A Configurable Input/Output (I/O) Block, shown in Figure 2.3, brings signals onto the chip and transmits them off again. This block comprises an input buffer and an output buffer, each featuring controls for three-state and open collector outputs. Typically, pull-up resistors are incorporated on the outputs, with the potential addition of pull-down resistors to terminate signals and buses internally, eliminating the need for external resistors. Additionally, users can program the output polarity as either active high or active low and adjust the slew rate for fast or slow rise and fall times. Incorporating flip-flops on outputs enables clocked signals to be output directly to the pins, reducing delay and facilitating compliance with setup time requirements for external devices. Similarly, flip-flops on inputs minimize the signal delay before reaching a flip-flop,

thereby reducing the FPGA's hold time requirement.

Digital Signal Processing (DSP) Slice

The DSP slice, block, or cell, often referred to interchangeably, represents a specialized unit within an FPGA dedicated to digital signal processing (DSP) tasks. Unlike general-purpose CLBs, which can handle various functions, the DSP slice is optimized explicitly for filtering, multiplying, and other DSP operations. This specialization allows it to execute these functions more efficiently and quickly than implementing them using multiple CLBs. Typically, DSP slices are equipped with dedicated hardware resources such as multipliers, accumulators, and specialized adders tailored to perform DSP computations rapidly and with minimal resource overhead. Their presence significantly enhances the FPGA's ability to handle real-time signal processing tasks, making them indispensable in applications ranging from telecommunications and audio to image and video processing [27].

Block Random Access Memory (BRAM)

Memory options on an FPGA board vary, but the dedicated memory on the chip itself is known as block RAM or BRAM. Each block has a fixed size, such as 36K bits for Xilinx 7 series chips, but these blocks can be divided or combined to create smaller or larger BRAM sizes. Additionally, they offer various operational settings and can support special features like error correction [28].

Clock Circuitry

The chip contains special I/O blocks equipped with powerful clock drivers that are designed to handle high-drive clock signals. These drivers connect to clock input pads and transmit the clock signals onto the global clock lines. These

global clock lines are optimized for minimal skew and rapid signal propagation. It's important to note that synchronous design is essential with FPGAs because only the global clock lines can ensure consistent skew and delay [29].

2.1.2 FPGA Hardware Design Flow

The flowchart for designing and implementing hardware on the FPGA is shown in Figure 2.4 [30]:

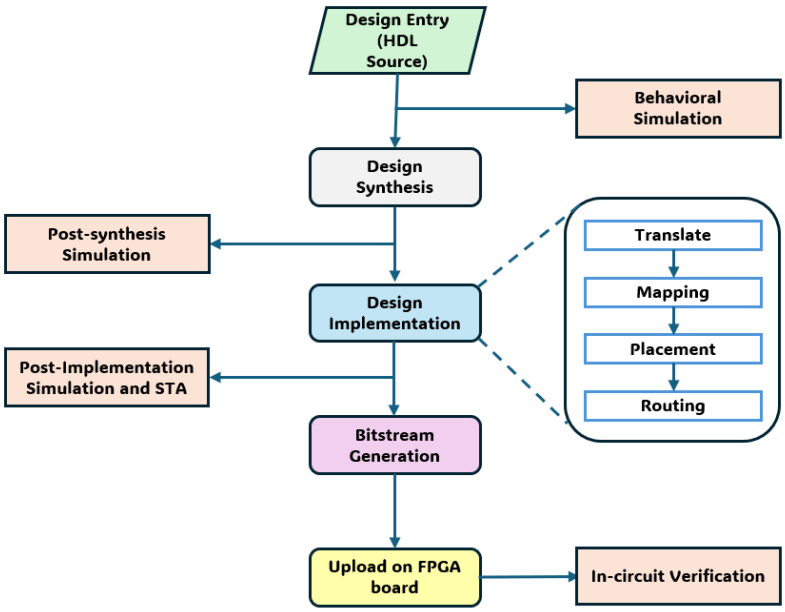


Figure 2.4: FPGA Hardware Design Flow.

Design Entry

The logic description can be created using a schematic editor, a finite state machine (FSM) editor, or a hardware description language (HDL). This involves selecting components from a library and directly mapping the design functions to chosen computing blocks. When managing designs with many functions becomes challenging visually, HDL can be utilized to capture the design struc-

turally or behaviorally. In addition to VHDL and Verilog, which are well-established HDLs, several C-like languages are available, such as Handel-C, Impulse C, and System C [31] [32] [33] .

Behavioral Simulation

This step plays a crucial role in ensuring the accuracy of the HDL by comparing the output of the HDL model with the behavioral model. Utilizing RTL description, behavioral simulation allows for functionality testing with the assistance of Electronic Design Automation (EDA) tools [34].

Design Synthesis

This process involves translating the HDL code into a device netlist format, representing a complete circuit with logical elements. Synthesis includes checking the code syntax, analyzing the hierarchy of the design architecture, and compiling the code along with optimization. The resulting netlist is saved as a .ngc file [35].

Design Implementation

The design implementation comprises the following steps [36]:

1. **Translate:** During translation, netlists are combined into a single NGD file, followed by timing and logical design rule checks. Constraints from the user constraint file (UCF) are then integrated into the merged netlist.

2. **Map:** In this phase, the tool assigns resources to basic logic elements and handles location and timing constraints. It then optimizes the target and produces the physical design database, along with a post-map STA report detailing block and routing delays.

3. **Place and Route:** This phase involves placement and routing of the design, resulting in the post-place-and-route STA report detailing all nets and delays in the design.

Device Programming

The routed design must be loaded and converted into a format supported by the FPGA. Thus, the routed .ncd file is provided to the BitGen program, which generates a bitstream file containing all programming information for the FPGA [37].

Timing Analysis

In this phase, a timing tool checks whether the implemented design meets user-specified timing constraints, including clock frequency, setup violation, and hold violation.

2.1.3 FPGA Software Development Flow (Xilinx SDK)

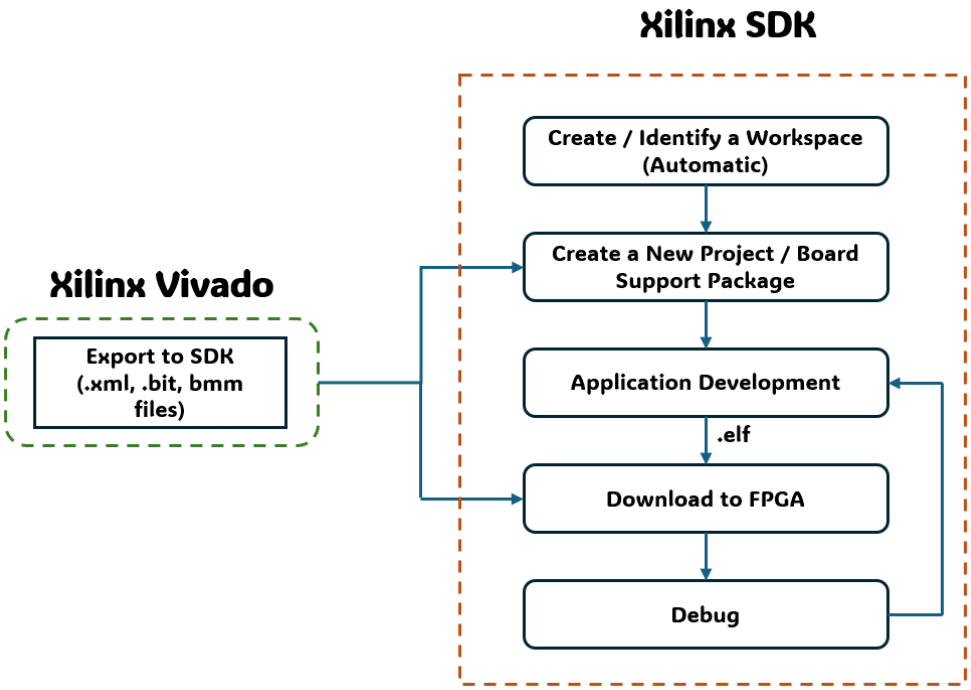


Figure 2.5: FPGA Software Development Flow (Xilinx SDK).

The typical process for developing a software application for Vivado® embedded system design using Xilinx® SDK is as follows:

1. Launch Xilinx SDK. When prompted, open an existing workspace or create a new one using a Hardware Platform generated from Vivado® IP Integrator.

2. Begin developing the software application. Xilinx SDK provides documentation for software libraries and drivers included in the board support package.

3. SDK automatically generates a default linker script for the application. Use linker generation tools to adjust the memory map as needed.

4. When ready to test the application on the hardware target, create a run/debug configuration to run/debug the application. If necessary, download the hardware bitstream to the FPGA device.

2.1.4 Advantages and Disadvantages of FPGA

FPGAs offer numerous advantages. They can execute faster and parallel signal processing, a task typically challenging for processors, as they can be programmed at the logic level. Unlike ASICs, which are fixed once programmed, FPGAs are reprogrammable at any time, even remotely, enabling multiple reuses. Their ready availability and swift programming using HDL code ensures faster solutions to market. Moreover, FPGA development is cost-effective, requiring fewer expenses and less costly tools than ASICs, eliminating significant Non-Recurring Expenses (NRE). Additionally, FPGA design involves minimal manual intervention, as software manages routing, placement, and timing, streamlining the design flow and eliminating complexities.

FPGAs have several drawbacks. Programming requires expertise in VHDL / Verilog and digital system fundamentals, unlike the more straightforward C programming used in processor-based hardware. Additionally, power consumption is higher, with limited control over optimization compared to ASICs. Once a specific FPGA is chosen, programmers are constrained by available resources, limiting design size and features. Choosing the right FPGA from the start is crucial. While suitable for prototyping and low-quantity production, costs increase with higher FPGA quantities, unlike ASICs.

2.1.5 Kintex-7 KC705 FPGA Board

Figure 2.6: Xilinx Kintex-7 KC705 FPGA Board.

The KC705 evaluation board for the Kintex®-7 FPGA offers a hardware environment suitable for developing and evaluating designs targeting the Kintex-7 XC7K325T-2FFG900C FPGA. It includes features commonly found in many embedded processing systems, such as DDR3 SODIMM memory, an 8-lane PCI Express® interface, a tri-mode Ethernet PHY, general-purpose I/O, and a UART interface. Additional features can be integrated using FPGA Mezzanine Cards (FMCs), which can be connected to either of the two VITA-57 FPGA mezzanine connectors provided on the board. These connectors support both High Pin Count (HPC) and Low Pin Count (LPC) FMCs. Additionally, the board includes features like an I2C bus, status LEDs, switches, and configuration ports. Figure 2.6 shows the KC705 FPGA board. A detailed explanation of the KC705 board is provided in [38].

2.2 Communication Protocols

The communication protocol plays a significant role in coordinating communication between devices. It is tailored differently depending on system needs, with specific rules agreed upon between devices to ensure successful communication. In heterogeneous computing systems, continual communication between FPGA and the host computer is essential. Various FPGA communication protocols are used for this purpose. Here, we delve into two of the most common: UART and Ethernet.

2.2.1 UART: Universal Asynchronous Receiver-Transmitter

UART is a fundamental communication protocol in various systems, including embedded systems, microcontrollers, and computers, primarily due to its simplicity and efficiency. Unlike other communication protocols, such as SPI or I2C, UART requires only two wires for transmitting and receiving data, making it a preferred choice in many applications.

UART operates on asynchronous serial communication at its core, offering configurable speeds to accommodate diverse system requirements. In this asynchronous mode, data transmission occurs without a synchronized clock signal between the transmitting and receiving devices. Instead, UART devices rely on predefined baud rates to ensure proper data transfer [39].

Each UART device comprises two essential signals: the Transmitter (Tx) and the Receiver (Rx). These signals enable the transmission and reception of serial data between devices, facilitating seamless communication. As shown in Figure 2.7, the transmitting UART initially sends data in parallel form over a controlling data bus, which is then serialized and transmitted to the receiving UART bit by bit. Here, the serial data is converted back into parallel form for the receiving device's use.

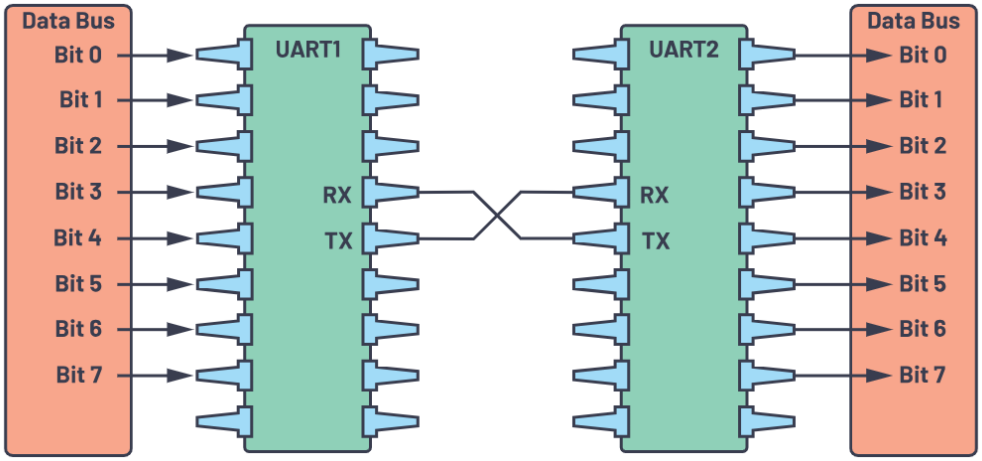


Figure 2.7: UART with Data Bus [39].

Despite its simplicity, synchronization is crucial in UART communication to ensure accurate data transmission and reception. The transmitting and receiving devices must operate at the same baud rate, with a permissible deviation of up to 10%. Beyond this threshold, discrepancies in timing may occur, leading to data handling issues during communication processes. Therefore, carefully considering baud rate alignment is essential for successful UART communication. The UART frame format is shown in Figure 2.8 [40].

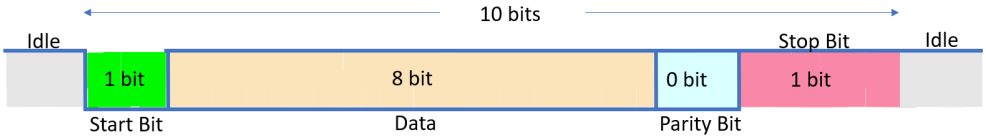


Figure 2.8: UART Frame Format.

Advantages and Disadvantages of UART Communication Protocol

UART communication presents several benefits [41]:

1. It operates efficiently by utilizing only two wires, simplifying the hardware setup required for communication between devices.

2. Unlike synchronous communication protocols, UART does not necessitate a dedicated clock signal, reducing system complexity.

3. UART incorporates a parity bit, enabling error checking to ensure data integrity during transmission.

4. The flexibility of UART allows for modifying the data packet structure as long as both transmitting and receiving ends are configured accordingly.

However, despite its advantages, UART communication comes with its limitations [41].

1. One notable drawback is the restricted size of the data frame, which is limited to a maximum of 9 bits. This limitation may pose challenges when dealing with larger data sets.

2. UART does not support multiple slave or master systems, limiting its applicability in specific scenarios where such configurations are necessary.

3. Maintaining consistent baud rates across UART devices is crucial, as deviations exceeding 10% may lead to synchronization issues and data transmission errors, posing challenges in ensuring reliable communication between devices.

2.2.2 Ethernet

As shown in Figure 2.9, the OSI networking model [42] [43] comprises seven layers, each with its own protocols, including the physical, data link, network, transport, session, presentation, and application layers. The first two layers, the physical and data link layers, are particularly associated with a widely used set of Ethernet protocols. Ethernet, defined by the IEEE 802.3 standard, is a prevalent networking technology utilized in various forms. It encompasses protocols,

ports, cables, and computer chips necessary to connect devices to a local area network (LAN) for efficient data transmission via coaxial or fiber optic cables.

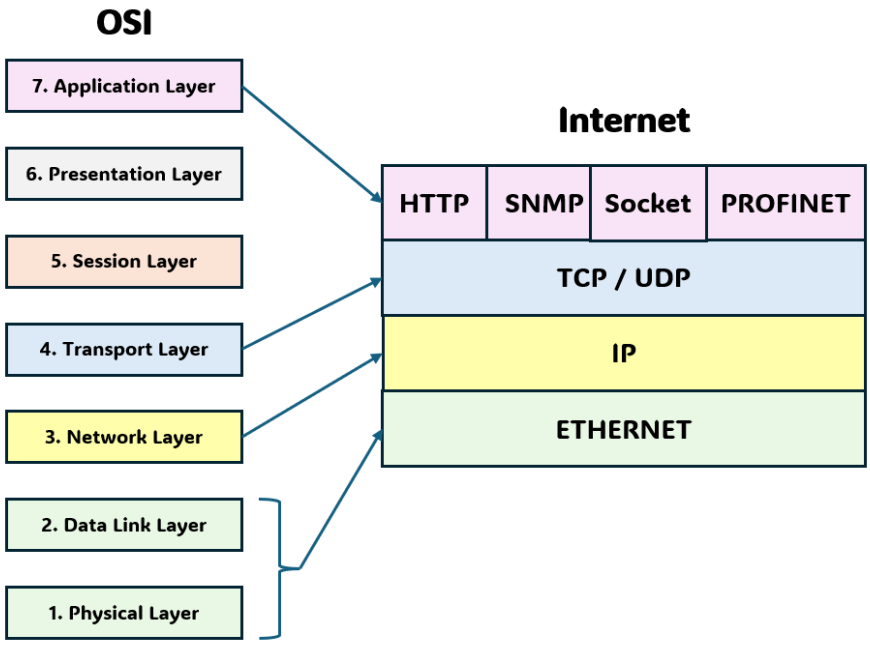


Figure 2.9: OSI Model.

Within the OSI network model, Ethernet protocol operates primarily at the first two layers: the Physical and Data Link layers. However, as shown in Figure 2.10, Ethernet further divides the Data Link layer into two distinct layers: the Logical Link Control (LLC) layer and the Medium Access Control (MAC) layer [44]. The physical layer focuses on hardware elements such as repeaters, cables, and network interface cards (NICs), specifying cable types, lengths, and optimal network topologies. Meanwhile, the data link layer manages data packet transmission between different nodes. Ethernet employs an access method known as CSMA/CD (Carrier Sense Multiple Access/Collision Detection), where computers listen to the cable before transmitting data to avoid collisions on the network.

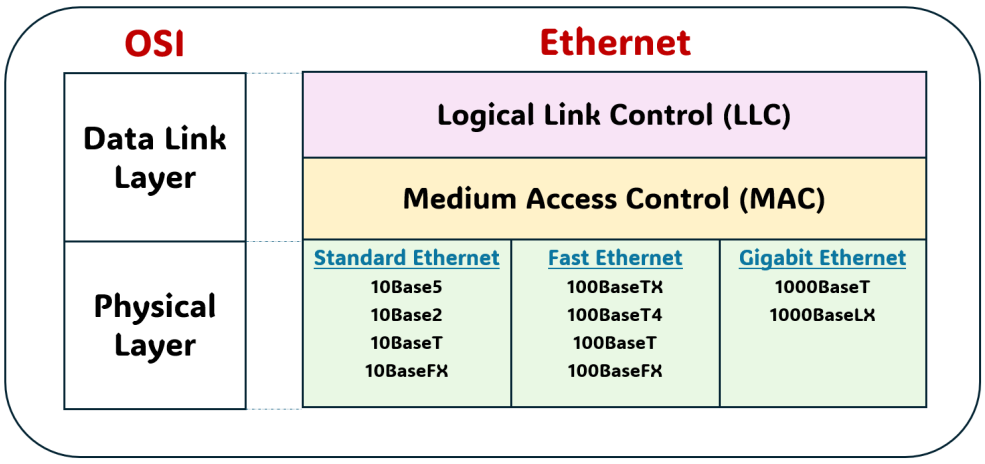


Figure 2.10: Ethernet Protocol Architecture.

Ethernet protocol transmission speeds are measured in Mbps (millions of bits per second) and are available in three types: 10 Mbps (Standard Ethernet), 100 Mbps (Fast Ethernet), and 1,000 Mbps (Gigabit Ethernet). The actual transmission speed represents the maximum achievable speed under ideal conditions, though real-world network output often needs to catch up to this highest speed.

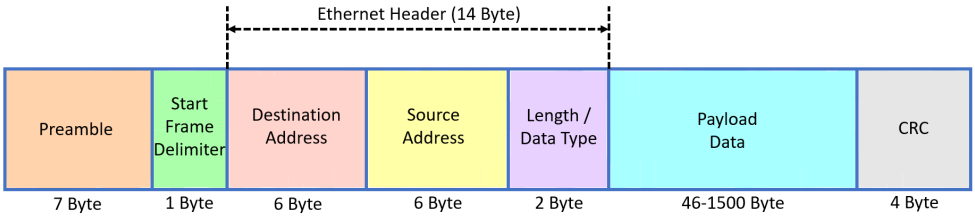


Figure 2.11: IEEE 802.3 Ethernet Frame Format.

As shown in Figure 2.11, the Ethernet frame format [45] required for all MAC implementation is defined in the IEEE 802.3 standard. It begins with the Preamble and SFD, both functioning at the physical layer. Following this, the Ethernet header contains both the Source and Destination MAC addresses, succeeded by the frame's payload. Lastly, the CRC field is included to detect

errors within the frame.

Ethernet communication has several advantages:

1. High speed: Ethernet offers faster speeds compared to wireless connections, up to 10Gbps or even 100Gbps.
2. Efficiency: Ethernet cables like Cat6 are power-efficient, consuming less power than Wi-Fi.
3. Good data transfer quality: Resilient to noise, ensuring high-quality data transfer.
4. Security: Provides higher security levels, controlling network access to prevent unauthorized access.
5. Low cost: Ethernet setup is inexpensive and requires minimal investment.
6. Reliability: Minimal radio frequency interruptions ensure reliable connections with no bandwidth shortages.

However, despite its advantages, Ethernet communication comes with its limitations.

1. Limited expandability: Additional expenses and time required for expanding networks due to rewiring and additional equipment.
2. Crosstalk with longer cables: Longer Ethernet cables can suffer from crosstalk.
3. Restricted connections: A single Ethernet connection allows for only one device; multiple devices require more cables.
4. Not ideal for real-time applications: Ethernet lacks deterministic services, making it unsuitable for real-time or interactive applications.

5. Limited mobility: Suitable for stationary devices, limiting mobility.

6. Inefficiency with traffic-intensive applications: Efficiency decreases as Ethernet traffic increases.

7. Complex installation: Requires professional assistance for installation.

8. Troubleshooting difficulties: Identifying problematic cables in the network can be challenging.

2.3 Floating Point Numbers Conversion in IEEE-754 Format

Floating point numbers are used in a wide range of applications, including high performance scientific computing, 3D graphics rendering, and financial analysis. To store any floating point number in a computer, mainly two IEEE representations are used [46].

1. Single Precision (32-bit) Format

2. Double Precision (64-bit) Format

The normalized form of the floating number is:
$(-1)^S \times 1.M \times 2^E$
Where: S = Sign, M = Mantissa and E = Exponent

2.3.1 Single Precision (32-bit) Format

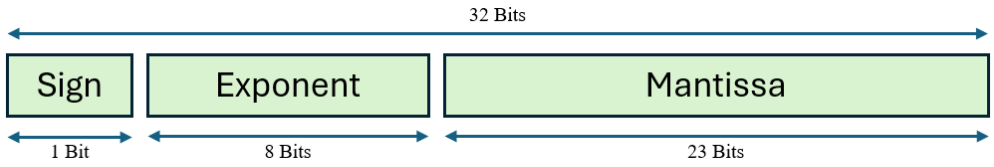


Figure 2.12: IEEE-754 Single Precision Format..

2.3.2 Double Precision (64-bit) Format

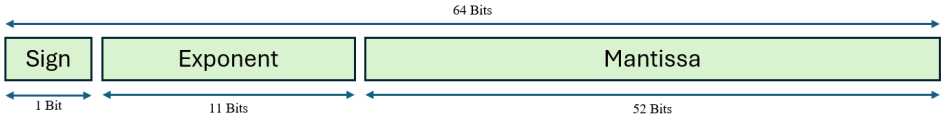


Figure 2.13: IEEE-754 Double Precision Format.

Chapter 3 Hardware-Software Co-design Development: Using UART or Ethernet

3.1 Introduction

As shown in Figure 3.1, the proposed system is divided into two parts to achieve heterogeneous computing functionality: 1. hardware, implemented on the FPGA, and 2. software, implemented on the host computer. The FPGA hardware handles complex, time-intensive computations while the host computer manages controlling functions and sequential computations. Compared to the PCIe mode, the PCIe communication protocol is replaced with UART or Ethernet communication protocols.

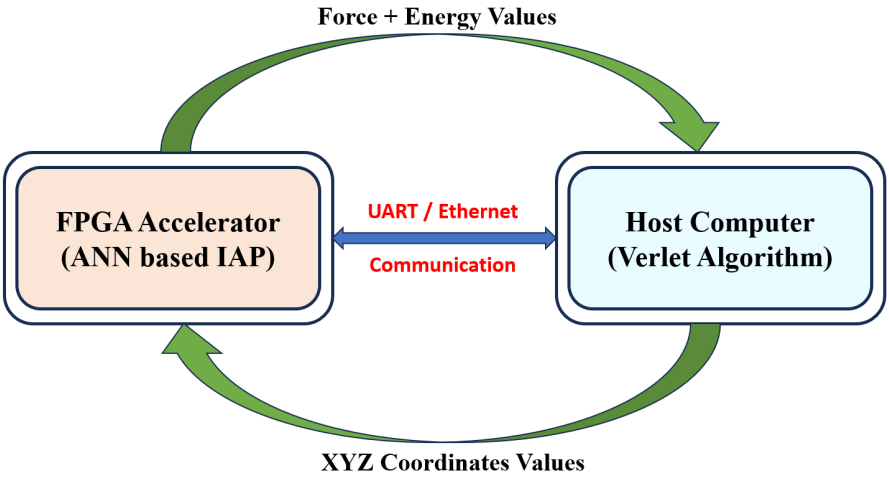


Figure 3.1: Proposed Hardware-Software Co-design Prototype for MD Simulation.

3.2 System Architecture Overview

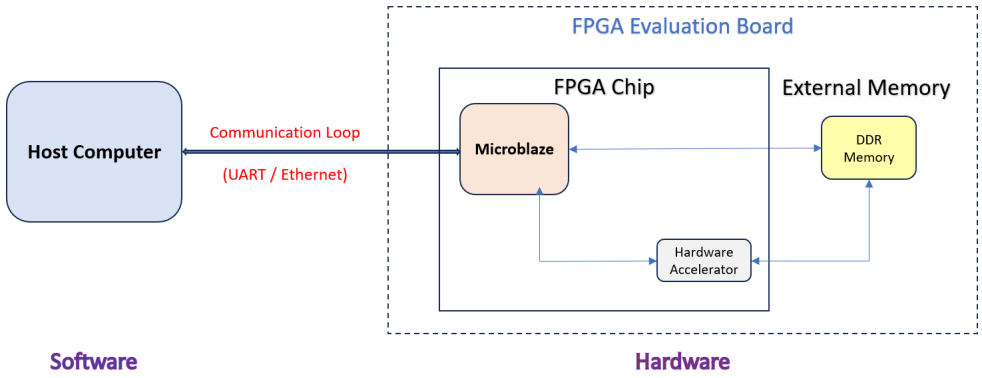


Figure 3.2: Block diagram of hardware-software co-design for MD simulation.

The computation is executed on the FPGA using digital IPs. Since most of the computations are data-intensive, there is constant data transfer between the hardware accelerator IP and memory. Once the computation is complete, the final data is transferred between the FPGA and the host computer via communication IPs (e.g., UARTLite or EthernetLite). Figure 3.2 illustrates two IPs: MicroBlaze and DDR memory. During computation, a large number of multi-dimensional arrays need to be stored for further processing, which is managed by the DDR memory controller. MicroBlaze, a soft-core processor, functions as the master controller of the hardware.

3.2.1 Xilinx FPGA IPs

Microblaze: Soft-Core Processor

It is a 32-bit general-purpose Reduced Instruction Set Computer (RISC) soft-core processor. This processor features a 32-bit general-purpose register, RISC Harward Architecture, a 3-stage pipeline, and an interrupt module [47]. Utilizing the local memory bus (LMB), Microblaze accesses on-chip memory and

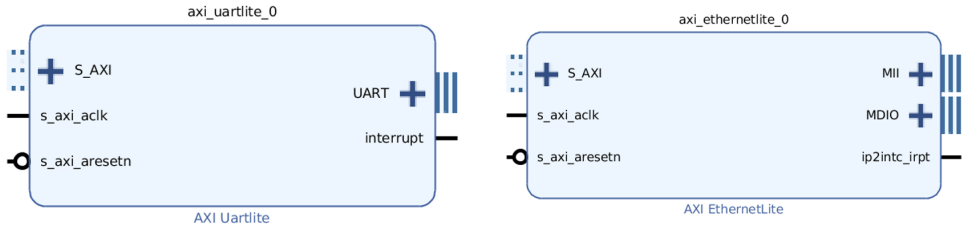
is compatible with the IEEE 754 single-precision floating-point format. It also includes an Instruction Cache (IC) and Data Cache (DC), exception handling, a debug module, and a barrel shifter. Except for the Zynq family, Microblaze is supported in most Xilinx FPGA families (Artix-7, Kintex-7, Spartan, etc.). A comprehensive explanation of Microblaze is provided in [48].

Figure 3.3: Microblaze IP.

Xilinx provides a software environment called Software Development Kit (SDK) with the Embedded processor (Microblaze). The SDK supports C/C++ languages for writing software code and is responsible for controlling the operation of the Microblaze soft-core processor. It extends support to all peripherals IPs used with the processor. Figure 3.3 shows the Microblaze IP.

Communication IP (AXI UARTLite / EthernetLite)

The Advanced eXtensible Interface (AXI) UARTLite serves as a control interface for asynchronous serial data transfer. It supports full-duplex communication, providing AXI4-Lite interface register access and data transfer. The receiver and transmitter (First in First out) FIFO size are limited to 16 Bytes. This module incorporates configurable baud rates (e.g., 9600, 115200, 421800, 921600, etc.). UARTLite IP follows the standard UART frame format. A comprehensive explanation of AXI UARTLite is available in [49]. The AXI UARTLite IP is shown in Figure 3.4 (a).



(a) **(b)**

Figure 3.4: (a)AXI UartLite IP and (b) AXI EthernetLite IP.

The AXI EthernetLite is designed to incorporate the features of the IEEE 802.3 Ethernet standard. It facilitates connection to external 10/100 Mb/s physical (PHY) transceivers through the Media Independent Interface (MII). It utilizes the AXI4/AXI4-Lite on-chip communication protocol to enable communication with the Microblaze soft-core Processor. EthernetLite IP follows the IEEE 802.3 standard Ethernet frame format for communication. A comprehensive explanation of the AXI EthernetLite is provided in [50]. The AXI EthernetLite IP is shown in Figure 3.4 (b).

Memory Interface Generator (MIG)

The core of the Xilinx® 7 series FPGAs memory interface solutions comprises a pre-engineered controller and physical layer (PHY). It facilitates interfacing 7 series FPGA user designs and AMBA® AXI4 slave interfaces with DDR3 and DDR2 SDRAM devices. Within the Embedded Development Kit (EDK), this core is accessible through the Xilinx Platform Studio (XPS) as the axi_7series_ddrx IP, featuring a static AXI4 to DDR3 or DDR2 SDRAM architecture. The Kintex-7 KC705 board contains DDR3 SDRAM, which is utilized with this MIG IP. A detailed explanation of the 7 series MIG is available in [51]. The MIG IP is shown in Figure 3.5.

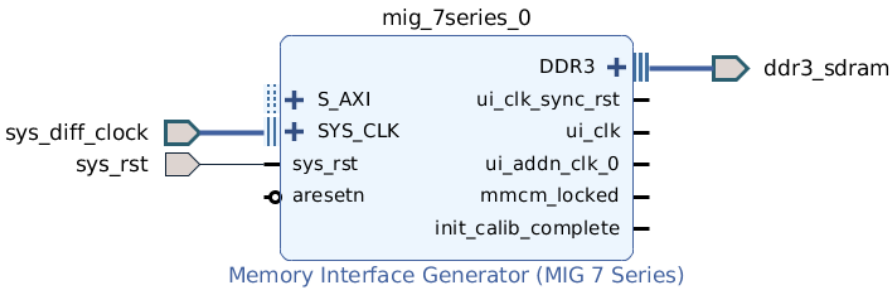


Figure 3.5: Memory Interface Generator IP.

AXI InterConnect

The AXI Interconnect core facilitates the connection of a mixture of AXI master and slave devices, allowing variations in data width, clock domain, and AXI sub-protocol (AXI4, AXI3, or AXI4-Lite). In instances where the interface characteristics of any connected master or slave device differ from those of the crossbar switch within the interconnect, the necessary conversions are automatically performed by inferring and connecting the appropriate infrastructure cores within the interconnect. A detailed explanation of the AXI Interconnect is available in [52].

3.2.2 Hardware Accelerator HLS IP for ANN-based IAP Calculation

In this subsection, FPGA hardware blocks are introduced for reconfigurable high performance computation. Generally, high performance computation contains multiple loops. So, in conventional processors, all loops run sequentially. However, FPGA can take up several independent serial loops and parallelize them [53]. So, when implemented on FPGA, similar tasks are performed concurrently, and the algorithm will be converted into multipliers and adders (MAC blocks) that go along multiple loops.

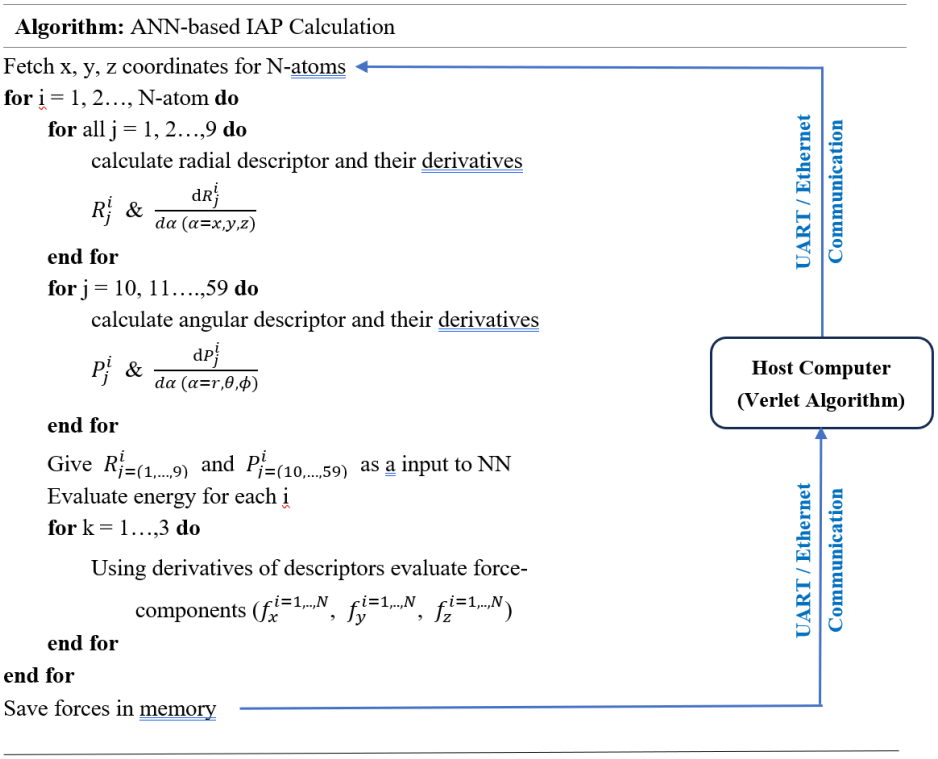


Figure 3.6: Algorithm for ANN-based IAP Calculation.

The algorithm for constructing the ANN-based IAP for gold nanoparticles, as described in [18], is illustrated in Figure 3.6. The first step involves obtaining the Cartesian coordinates of the atomic placements, which are then transferred from the host computer to the FPGA via UART or Ethernet communication. These coordinates are stored in memory. The algorithm that uses these coordinates to calculate the forces and energy is performed on the FPGA and is implemented as an IP (Intellectual Property) block within the FPGA design.

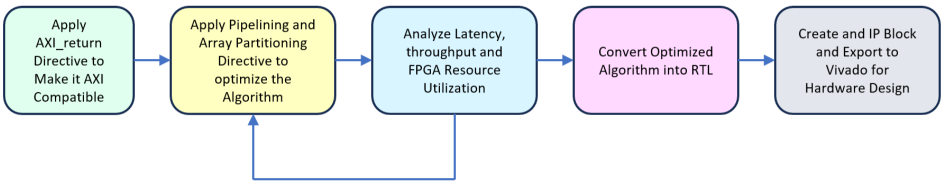


Figure 3.7: Design Flow for HLS IP Creation.

Vivado-HLS (High-Level Synthesis) software was chosen to accelerate this high performance computation algorithm. Utilizing the C-based code of the above-described algorithm in Vivado HLS, it was converted into HDL. As shown in Figure 3.7, the AXI_return directive made it AXI-compatible after composing the C language code. After that, a pipeline directive was applied to improve the latency and throughput of the system. Also, all operations are floating-point operations and contain multidimensional arrays. So, it requires significant space in memory. Array partitioning directive is used to optimize multidimensional arrays. After achieving the optimal design, the algorithm was exported to the RTL design and converted into IP. This module's physics and mathematical calculations are completely discussed in Bulusu et al. [18].

3.3 Hardware-Software Co-design Implementation

The data flow direction is shown in Figure 3.8. The inputs of the proposed system consist of the (X, Y, Z) Cartesian coordinates of atoms, while the outputs include forces and total energy of the cluster. Since MD simulation is an iterative process, a separate coordinates file is used to initialize the system. Initially, the Cartesian coordinates are sent from the computer to the FPGA using the software C-language code that we developed, which we will explain in detail in the particular UART or Ethernet implementation. Following initialization, the MD simulation performs calculations and generates forces and total energy for the atomic cluster.

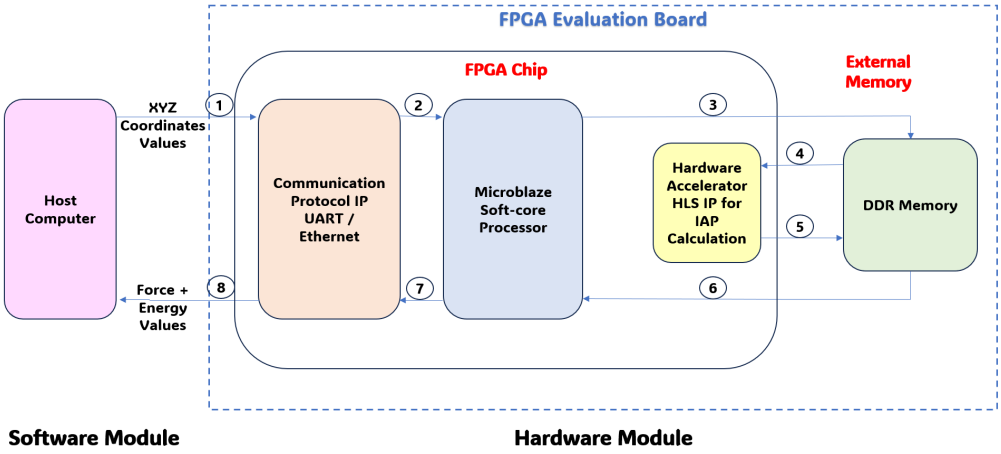


Figure 3.8: Data Flow Diagram of System Implementation for MD Simulation.

On the FPGA side, coordinates are received using UART/Ethernet IP and stored in DDR memory with the help of the MicroBlaze microcontroller. Subsequently, the HLS IP performs calculations and generates outputs in terms of atomic forces and total system potential energy, which are stored in DDR memory. These outputs are then sent back to the host computer. Upon receiving force and energy values from the FPGA board, the next cycle's coordinates are generated by processing the previous cycle's force values on the host computer using Verlet algorithm. Implementations were carried out for Au_{147} and Au_{309} atomic nanoclusters. The data transfer size varies according to the cluster size.

The complete FPGA implementations of UART and Ethernet communication-based heterogeneous computing systems are discussed in detail in the following subsection.

3.3.1 UART Implementation

On FPGA board (Hardware Module)

To design a communication interface, a schematic-based design was developed in Xilinx Vivado [54]. This design included DDR3 SDRAM on the Kintex-7

KC-705 board for data storage during processing. The Microblaze soft-core processor acted as the master controller, operating at 300 MHz, while the hardware accelerator MD simulation IP operated at 100 MHz. Communication between the FPGA and the host computer was handled by the AXI UartLite IP. Other Peripheral IPs communicated with the Microblaze processor through AXI Interconnect or SmartConnect IP.

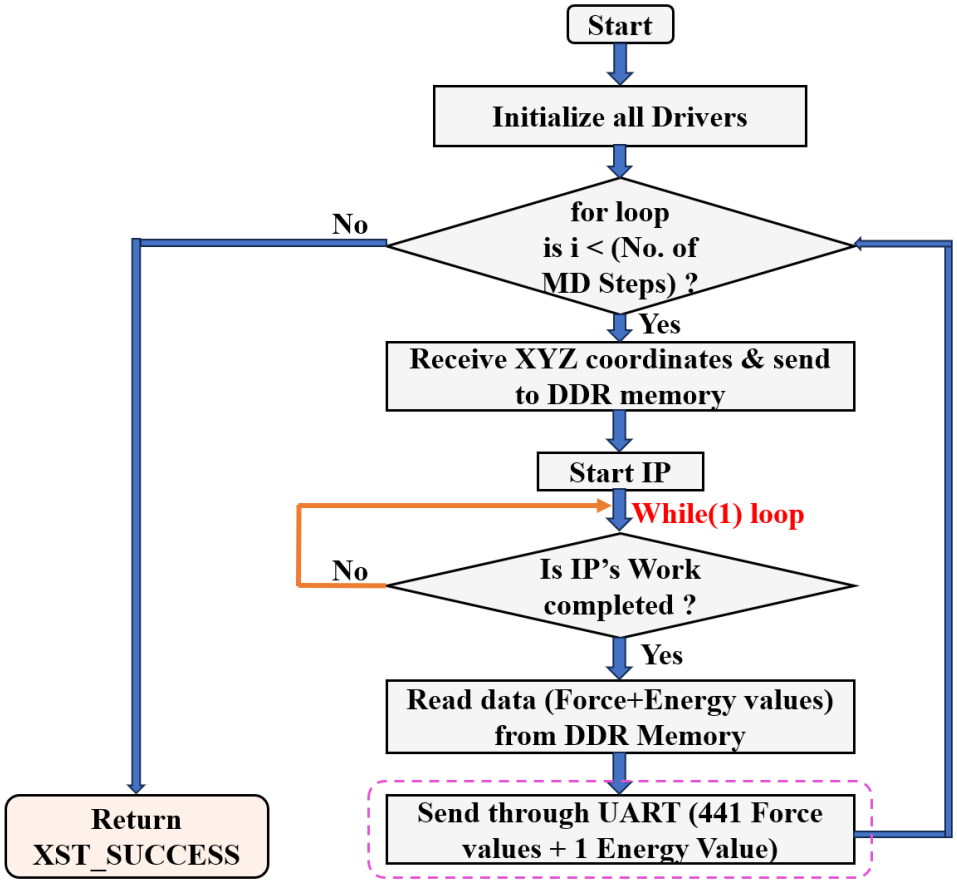


Figure 3.9: SDK Flow Diagram for UART Implementation.

An Embedded-C language code was written in Xilinx SDK to control the hardware [55]. This code handled tasks such as data reception and transmission, data storage in DDR memory, and initiation and termination of the hardware accelerator module. The flowchart of the SDK code is shown in Figure 3.9.

The structure of the SDK code is nearly identical for UART and Ethernet-based designs. Changes related to communication interfaces were highlighted with different colored rectangular dotted boxes.

As shown in the flowchart in Figure 3.9, all IP drivers utilized in the block diagram were initiated, and a for loop was introduced to define the number of MD cycles. Using a while loop, XYZ coordinates were awaited from the host computer. After receiving the coordinates, they were stored in DDR memory as specified. Then, the hardware accelerator module was started, using a polling mechanism to check if the IP's calculation was complete. When the IP's work was finished, force and energy values were automatically stored in DDR memory at the specified location. The SDK coding procedure remained the same for UART and Ethernet-based systems before the output data was sent back to the host computer. The output data was sent back using the UartLite standard SDK function for UART-based designs. The UartLite IP used the standard frame format for sending and receiving data through the UART interface. The standard UART frame format is discussed above.

On Host Computer C-code (Software Module)

The Verlet algorithm was used to execute the remaining sequential operations and compile all necessary MD files in the host computer. On the host computer side, a C-language code was used for UART-based communication to transmit XYZ coordinates from the host computer to the FPGA and to receive force and energy values from the FPGA. This code manages control and data transfer from the host computer. Algorithm 2 of the host computer software module for UART is shown in Figure 3.10.

Algorithm 2: Host computer Software Module (C - code) for UART

- Include all the Header files
- Start **main function**
  - Call UART interface definition function
  - Open UART interface
  - Call UART interface Send data function
  - Call UART interface receive & store data function
  - Close UART interface
- end **main Function**
- Start **Interface definition function**
  - Set all the parameters (e.g. Baud rate, parity, start and stop bit) as per UART interface
- end **Interface definition function**
- Start **Send data function**
  - Read data from input file
  - Data send
  - Check condition to complete data sending
- end **Send data function**
- Start **Receive data function**
  - Open output file
  - Receive data
  - Check condition to complete data sending
  - Convert receive data into floating number
  - Store floating numbers into output file
- end **Receive data function**

Figure 3.10: Algorithm of host computer software Module (C-code) for UART.

In the main function, all the functions are called step by step. The first function called was to define the UART interface port for connectivity. Following that, the send data function was invoked, which reads data from the input files and sends it to the FPGA via the UART interface. Subsequently, the program waited for the output data received from the FPGA, and once the data was received, it was stored in floating-point format.

To establish the UART port connection, the definition of the UART interface is outlined in the interface definition function. This function defines the UART baud rate, parity bit, start and stop bits, and the size of data bits. In our design,

a baud rate of 921600 is used.

Once the UART connection is established, the UART send data function is invoked. Initially, this function opens the input coordinates file and reads data from it. Subsequently, the data is organized in UART frame format, transmitted via UART, and finally, the file is closed.

Following the successful completion of data transmission, the receive data function is triggered. Within this function, the output binary file is initially generated, followed by waiting for the complete data reception. When the data reception is successfully completed, the store float data function is executed.

The final step involves invoking the store float function to convert the received data into 32-bit floating point numbers. During the conversion process, each 32-bit data is grouped and stored in the file. This data is crucial for generating new Cartesian coordinates in the subsequent cycle.

3.3.2 Ethernet Implementation

On FPGA board (Hardware Module)

In the UART-based implementation discussion, a similar approach was followed for the Ethernet-based design. Firstly, a schematic-based design was developed in Xilinx Vivado [54]. The schematic design mirrors the UART-based design, the only difference being the communication interface IP. Compared to the UART-based design, the AXI UartLite IP is replaced by the AXI EthernetLite IP for Ethernet communication. This IP communicates with the Microblaze master using the AXI4 protocol and utilizes the Media Independent Interface (MII) for external communication with the host computer. Like the UART-based hardware design in Xilinx Vivado, Microblaze and its peripherals operate at 300 MHz, while our Hardware Accelerator MD simulation IP operates at 100 MHz.

The primary change in the Ethernet-based system is in the control aspects, where an Embedded-C language code is written in the Xilinx SDK [55]. The flowchart of the SDK code for Ethernet-based design is shown in Figure 3.11. As the UART implementation explains, the initial steps remain the same for both UART and Ethernet communication-based designs. The only difference occurs when sending output data back to the host computer, as indicated in Figure 3.11 by the green dotted rectangular box. After reading the output force and energy values, Ethernet frames are created according to the Ethernet standard IEEE 802.3. These Ethernet frames are then transmitted back to the host computer by leveraging the AXI EthernetLite driver's high-level functions in the SDK.

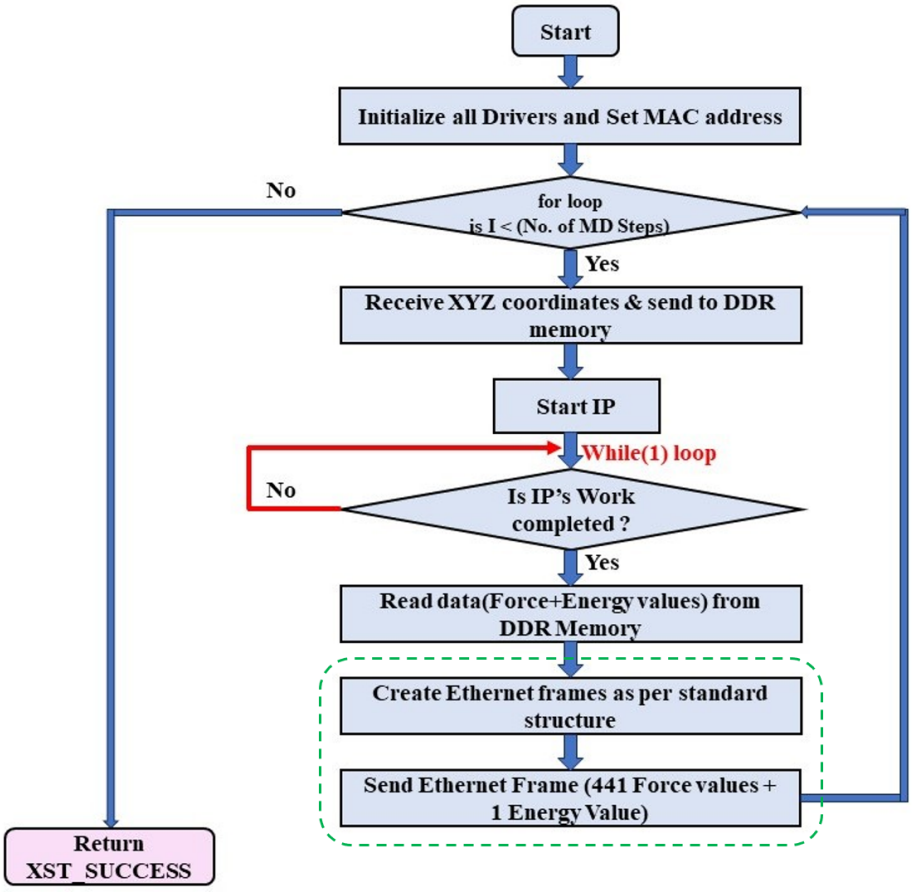


Figure 3.11: SDK Flow Diagram for Ethernet Implementation.

On Host Computer (Software Module)

The reception and transmission of Ethernet frames in the host computer are managed by C-language code. The Verlet software code remains identical for both UART and Ethernet communication, with the only difference being invocating the C language code for transmission and reception. The algorithm of this Ethernet's C-language code is shown in Figure 3.12.

Algorithm 3: Host computer Software Module (C - code) for Ethernet

- Include all the Header files
- Start **main function**
  - Open input file
  - Read input data (X, Y & Z Coordinates) from file
  - Set the length of frames
  - Assign destination & source MAC addresses
  - Create Ethernet frames as per standard IEEE 802.3
  - Send Ethernet frames
  - Open output file
  - Add filters to receive frames
  - Receive Ethernet frames
  - Trimmed overhead from received Ethernet frames
  - Convert payload data into floating numbers
  - Store floating numbers into output files
- end **main Function**

Figure 3.12: Host Computer Ethernet C-code for Open input file.

Initially, the input file was opened to read the Cartesian coordinate values. Within the input files, our input Cartesian coordinates are stored.

After opening the input file, the data from the file was read. The size of

the input data was 1768 bytes, comprising 147 floating-point values in each X, Y, and Z direction, along with one value for the MD cycle count. Given that the maximum payload size for Standard Ethernet is 1500 bytes, two Ethernet frames were necessary. Consequently, the data was divided into frames, with one containing a payload of 1000 bytes and the other 768 bytes.

Once the data reading process is finished, the Ethernet frame header following the IEEE 802.3 standard is created. Initially, the length of the payload data is defined, determining the total frame length. Next, the Ethernet frame type is determined, followed by fixing the source and destination MAC addresses according to our requirements. Since the input data is sent from the host computer to the FPGA, the source MAC address is the computer's address, while the destination MAC address is the FPGA's address.

Following the creation of Ethernet frames, each frame is compiled into standard Ethernet frame format and sent individually. The total length of the first frame was 1014 bytes (comprising a 14-byte frame header and 1000 bytes of payload data), while the size of the second frame was 782 bytes (consisting of a 14-byte frame header and 768 bytes of payload data).

After completing the data transmission, an output file was generated, and the system awaited the incoming data. For data reception, only data from the FPGA board was desired, so a filter was employed to block frames with other MAC addresses, allowing only those with matching source and destination MAC addresses to be received. In contrast to the input process, the source and destination MAC addresses were swapped in the output.

Similar to the input data, the output data was received in two frames. To await the arrival of both frames, a while loop was utilized. Upon receiving a frame, the payload data was extracted by discarding the initial 14 bytes of the frame header. Subsequently, the payload data was stored in the output file.

3.3.3 Data Transfer Comparision between Au_{147} and Au_{309}

Table 3.1: Data Transfer Comparision between Au_{147} and Au_{309}

Description (w.r.t per cycle)	**Au_{147}**	**Au_{309}**
No. of Bytes transfer (PC -> FPGA)	1768	3712
No. of Bytes transfer (FPGA -> PC)	1768	3712
No. of Ethernet Frames (PC -> FPGA)	2	3
No. of Ethernet Frames (FPGA -> PC)	2	3

Chapter 4
Results and Discussion

For performing the MD simulation, two different examples are explored. They are Au147 and Au309 nanoparticles. The FPGA used was Xilinx Kintex-7 KC-705 evaluation FPGA board. In addition to the accelerator IP developed, a Microblaze-based microcontroller system was also implemented through the block design of the Xilinx Vivado software. The accelerator outputs energy and force values which need to be transferred to the computer. A separate Verlet algorithm will take the force and energy values and calculate the next set of co-ordinates at the computer side. The accelerator IP on the FPGA works at 100 MHz, while the Microblaze processor and its peripherals operate at 300 MHz. For communicating between the FPGA and the computer, AXI UARTLite and AXI EthernetLite IPs were implemented. Microblaze processor controls the data transfer between the FPGA memory (RAM) and the different communication IPs, and this program was written in embedded C.

4.1 Results

4.1.1 Potential Energy Comparision

The first step in this exploration is to identify the accuracy of the results obtained through the proposed communication strategies. The calculated energy for every MD cycle from the FPGA-based heterogeneous computing system should match with the conventional approach (using a server). To check the accuracy, both UART and Ethernet-based communication were implemented, and the MD simulations were run on the board for 500 cycles. The results of energy and forces obtained at every cycle were compared with a conventional approach, and the results are plotted in Figure 4.1 and 4.2. There is no difference between the results obtained from all the approaches, and this proves that the proposed approach yields accurate results for several cycles as well. It validates that the total potential energy obtained using UART and Ethernet designs is correct and

identical to the PCIe-based design and HPC server, as explained in Bulusu et al. [18].

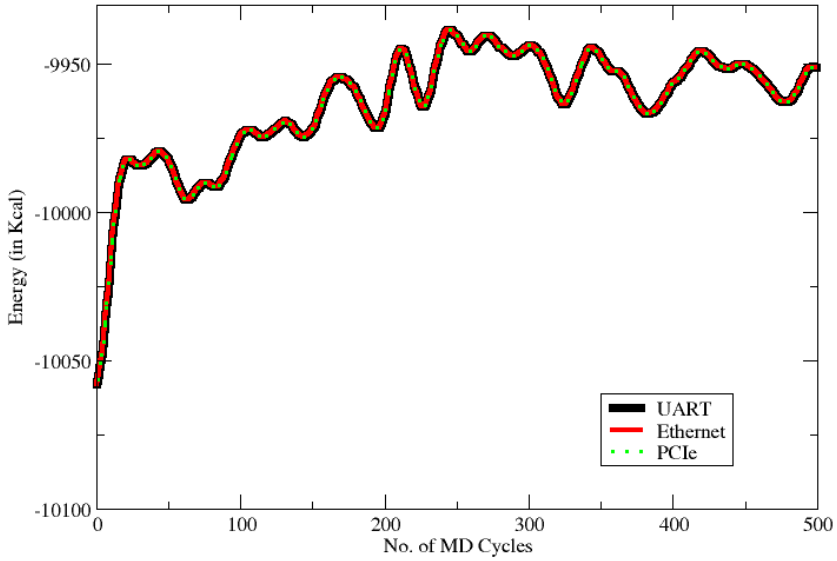


Figure 4.1: Potential Energy of Au_{147} vs. Number of MD Cycles.

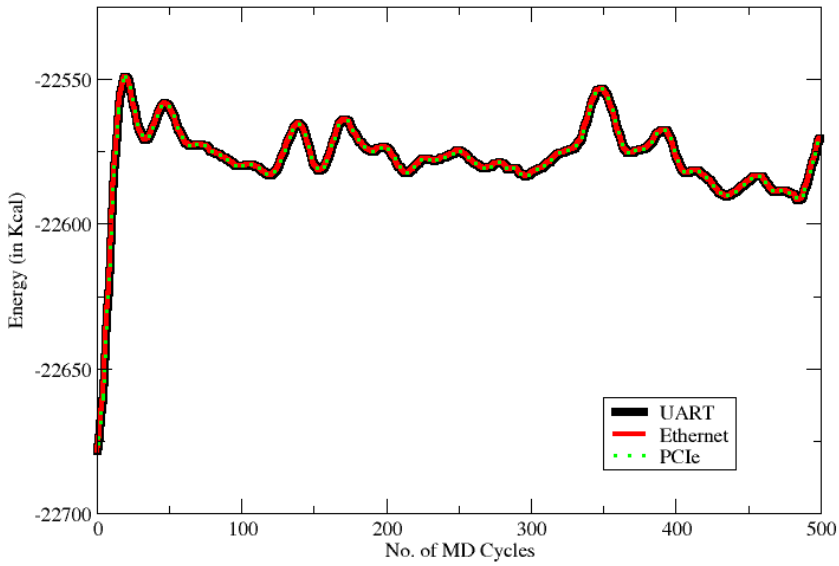


Figure 4.2: Potential Energy of Au_{309} vs. Number of MD Cycles.

4.1.2 Computation Time Comparision

Now that the accuracy of the system is verified, the system was explored to determine the computation time for the two different communication protocols implemented onto the FPGA board. A comparison of the computation time of the proposed two communication approaches with the conventional PCIe and HPC server is shown in Table 4.1 and Table 4.2 for Au_{147} and Au_{309} nanoparticles respectively. UART operates at a baud rate of 921600 bits/sec, while Ethernet works at a standard speed of 100 Mb/sec. PCIe speed is noted as 5 GT/sec, and in the HPC server, seven CPUs run in parallel at a frequency of 2600 MHz. Interestingly, UART and Ethernet-based communication protocols can withstand 50,000 iterations of MD computations, and the time it takes for 50,000 iterations comes to 17.54 hours & 18.7 hours for UART & Ethernet communication, respectively for Au147 and 64.36 hours & 65.56 hours respectively for Au309. This shows the robustness of the system and the potential to provide a lab-on-a-chip solution for such IAP computations.

Table 4.1: Computation Time Comparision of UART, Ethernet & PCIe Communication based Design for Au_{147}

No. of MD Cycles	**Computation Time**			
	UART	**Ethernet**	**PCIe**	**HPC Server**
	921600 bits/sec	100 MB/sec	5 GT/sec	—
1	1.26 sec	1.35 sec	1.39 sec	2.09 sec
10	13.89 sec	14.71 sec	15.18 sec	22.94 sec
100	2.12 min	2.26 min	2.34 min	3.51 min
500	10.54 min	11.20 min	11.58 min	17.43 min
5000	1.75 hr	1.86 hr	1.98 hr	2.90 hr
50000	17.54 hr	18.70 hr	19.84 hr	29.00 hr

Table 4.2: Computation Time Comparision of UART, Ethernet & PCIe Communication based Design for Au_{309}

No. of MD Cycles	**Computation Time**			
	UART	**Ethernet**	**PCIe**	**HPC Server**
	921600 bits/sec	100 MB/sec	5 GT/sec	—
1	4.74 sec	4.76 sec	4.83 sec	8.38 sec
10	52.19 sec	52.21 sec	53.57 sec	92.00 sec
100	7.99 min	8.01 min	8.12 min	14.01 min
500	39.62 min	39.72 min	40.25 min	69.94 min
5000	6.55 hr	6.56 hr	6.70 hr	11.64 hr
50000	64.36 hr	65.56 hr	66.97 hr	116.34 hr

Initially, XYZ values are sent to the FPGA, and the return data will be the computed force and energy values. For the Au147 atoms, 442 32-bit numbers (147 force values in each X, Y, and Z direction + 1 total energy value) were transmitted from the memory to the computer. The return communication was 441 32-bit numbers back to the FPGA. In such cases, the total data transfer would be 3.5 KB per cycle. Even though PCIe communication is faster, the overheads (e.g. frame construction) and loading of the drivers take more time. It is interesting to note that UART and Ethernet communication can achieve better computational efficiency compared to PCIe communication, despite PCIe being faster.

4.1.3 FPGA Resources Utilization

The next step is to determine the resources used by the FPGA board in both communication approaches. This has a direct correlation to the power of the

system. The results, as tabulated in Table 4.3, reveal that the UART and Ethernet designs, leveraging a Microblaze soft-core processor, demonstrate lower overall resource utilization than the PCIe-based design.

Table 4.3: UART, Ethernet & PCIe Communication based FPGA Implementation Resources Utilization for Au_{147}

FPGA Resources (%)	**UART**	**Ethernet**	**PCIe**
LUT	55.50	55.80	65.85
LUT RAM	19.34	19.36	23.58
Flip-Flop	26.34	26.87	32.44
BRAM	50.79	51.69	42.02
DSP Slices	49.40	49.40	49.52

4.1.4 On-Chip Power Consumption

For a Lab-on-a-chip application, power consumption plays a very important role. Table 4.4 shows the power consumed by the different communication protocols. It is clear that UART consumes the least power, followed by the other protocols. This again can be attributed to the simplicity of the communication protocols.

Table 4.4: On-Chip Power Comparision of UART, Ethernet & PCIe Communication based Design for Au_{147} (Au_{309})

Communication Protocols	**UART**	**Ethernet**	**PCIe**
Power (W)	5.9 (6.1)	6.0 (6.3)	8.9 (9.2)

4.2 Discussion

The communication speeds of UART and Ethernet 100×10^6 bps are three orders, respectively, and the same three orders of difference are observed between ethernet and PCIe 31×10^9 bps. Despite such high communication speeds, the computational time obtained from our experimental systems is quite contradictory and worth discussing and reasoning them.

While the packet size of UART is 10 bits of communication (8 bits of actual data), ethernet and PCIe communication communicates with several initial frames and library initialization. The reason behind such a contradictory result obtained could be ascertained by estimating the overheads of this communication protocol. Needless to say, UART has minimal overhead and can be considered negligible compared to its own communication speed.

Ideally, one complete cycle of the MD calculation with any communication protocol is a combination of computation and communication time. This is shown in Equation 4.1 below.

$$T_{(1\text{ cycle})} = T_{(\text{comm})} + T_{(\text{comp})} + T_{(\text{overhead})} \tag{4.1}$$

Where $T_{(1\text{ cycle})}$ is the total time taken for one complete cycle, $T_{(comm)}$ is the communication time and $T_{(comp)}$ is the time taken for computations in both FPGA And PC. $T_{(overhead)}$ is the overhead time required for the initialization of several libraries and frames, especially in the case of Ethernet and PCIe. $T_{(overhead)}$ for UART is negligible and therefore we will ignore its contribution in calculating $T_{(1\ cycle)}$.

4.2.1 Au_{147} Time Calculations

In the case of UART Communication for 1 cycle,

$$T^{U}{}_{(1\text{ cycle})} = 1.26 \text{ seconds (Table 4.1)} \tag{4.2}$$

$$T^{U}{}_{(\text{comm})} = \frac{\text{No. of bits transferred}}{\text{Speed/baudrate}} = \frac{35360}{921600} = 0.038 \text{ seconds} \tag{4.3}$$

From equation (4.1),

$$T_{(\text{comp})} = 1.22 \text{ seconds} \tag{4.4}$$

$T_{(comp)}$ is identical for all the communication protocols. $T^{U}{}_{(1\ cycle)}$ and $T^{U}{}_{(comm)}8$ - are the total time for 1 cycle and the communication time for UART communication protocol.

In the case of Ethernet Communication for 1 cycle,

$$T^{E}{}_{(1\text{ cycle})} = 1.35 \text{ seconds} \tag{4.5}$$

$$T^{E}{}_{(\text{comm})} = \frac{\text{No. of bits transferred}}{\text{Speed}} = \frac{28864}{100 \times 10^{6}} = 0.288 \times 10^{-3} \text{ seconds} \tag{4.6}$$

From equation 4.1, we know that

$$T^{E}{}_{(1\text{ cycle})} = T^{E}{}_{(\text{comm})} + T_{(\text{comp})} + T^{E}{}_{(\text{overhead})} \tag{4.7}$$

$$T^{E}{}_{(\text{overhead})} = 0.128 \text{ seconds} \tag{4.8}$$

$$T^{E}{}_{(overhead)} \ / \ frame \ = \ \frac{T^{E}{}_{(overhead)}}{No.\ of\ Ethernet\ frames} = \frac{0.128}{4} = 0.032 seconds \tag{4.9}$$

$T^{E}{}_{(1\ cycle)}$, $T^{E}{}_{(comm)}$ and $T^{E}{}_{(overhead)}$ are the total time for 1 cycle, communication time and the overhead time in case of Ethernet communication protocol.

From the above analogy, it is very clear that to find the tipping point where Ethernet communication will have a better hand compared to UART, the accountability of overhead of 0.032 seconds is to be taken into account. With these overheads in mind, one can propose which communication protocol should be used and can be predicted with simple calculations. The tipping point occurs when the time taken in 1 cycle using UART is the same as the time taken in 1 cycle using Ethernet in transferring some data. This can be shown as,

$$T^{U}{}_{(1\ cycle)} = T^{E}{}_{(1\ cycle)} \tag{4.10}$$

$$T^{U}{}_{(comm)} = T^{E}{}_{(comm)} + T^{E}{}_{(overhead)} \tag{4.11}$$

In equation 4.11, we omitted $T_{(comp)}$ because it is the same both for UART and Ethernet. Using equation 4.11, we can predict the tipping point by calculating $T^{U}{}_{(comm)}$ and $T^{E}{}_{(comm)}$ for different nanoparticles (number of bits transferred varies with the nanoparticles size).

To explain it experimentally, Table 4.5 shows $T_{(comm)}$ + $T_{(overhead)}$ time for different sizes of nanoparticles. It is evident from Table 4.7 that, for nanoparticles of approximately 430 atoms, the $T_{(comm)}$ + $T_{(overhead)}$ times for UART and Ethernet are almost the same. However, when the size exceeds 430 atoms, $T^{U}{}_{(comm)}$ time surpasses $T^{E}{}_{(comm)}$ + $T^{E}{}_{(overhead)}$ time resulting in better communication efficiency for Ethernet. The same analogy can be extended to PCIe communication as well. PCIe operates at a speed of 5 GTps (equivalent to 31 Gbps for a Gen2 8-lane PCIe bus). When $T^{E}{}_{(comm)}$ + $T^{E}{}_{(overhead)}$ reaches to 0.2 seconds, it equals to the $T^{P}{}_{(comm)}$ + $T^{P}{}_{(overhead)}$ time of PCIe communication. Estimated calculations suggest that this occurs when 3.5 million bits are transferred per cycle, which approximately corresponds to a nanoparticle size of 18,000 atoms. Practically, this size does not fall under the category of nanoparticles. Therefore, for MD calculations of nanoparticles or nanoalloys, Ethernet

communication offers better efficiency.

Table 4.5: Tipping Point Calculation for Different Protocols

Size of Nanoparticles	**No. of bits per cycle**	**Type of Communication**	$T_{(\text{comm})} + T_{(\text{overhead})}$ **in seconds**		
			1 Cycle	10 Cycles	100 Cycles
55 atoms	10624	UART	0.024	0.235	2.329
		Ethernet	0.117	1.179	12.082
		PCIe	0.174	1.79	17.775
147 atoms	28288	UART	0.054	0.515	5.138
		Ethernet	0.133	1.335	12.096
		PCIe	0.178	1.771	18.160
309 atoms	59392	UART	0.103	1.017	10.103
		Ethernet	0.130	1.280	12.120
		PCIe	0.183	1.856	18.670
430 atoms	82624	UART	0.133	1.346	13.512
		Ethernet	0.133	1.351	13.426
		PCIe	0.188	1.863	18.810
561 atoms	107776	UART	0.180	1.811	18.085
		Ethernet	0.136	1.374	13.965
		PCIe	0.197	1.838	18.209
923 atoms	177280	UART	0.293	2.933	29.409
		Ethernet	0.137	1.375	14.023
		PCIe	0.200	1.986	19.110

4.3 Conclusion

The ANN-based MD Simulation for Au_{147} and Au_{309} were implemented on the Xilinx Kintex-7 KC705 evaluation FPGA board. A hardware accelerator module with new communication strategies is proposed and implemented for 50,000 MD cycles. The computation time for 50,000 MD cycles is 17.54 hours and 18.7 hours for UART and Ethernet communication, respectively for Au147 and 64.36 hours & 65.56 hours respectively for Au309. Compared to the conventional HPC server, the proposed methodology has improved the computation time by 1.65 (1.81) times in UART and 1.55 (1.77) times in Ethernet communication for Au_{147} (Au_{309}) nanoparticles. The actual MD simulation requires more than 1 million cycles, so this computation time difference becomes more significant. The proposed systems significantly reduce resource utilization, resulting in decreased on-chip power consumption. In the implemented system, on-chip power consumption measured from Xilinx Vivado was 5.9 (6.1) Watts for UART and 6.0 (6.5) Watts for Ethernet, respectively, for Au_{147} (Au_{309}). Compared to conventional PCIe, on-chip power is reduced by 33% and 32% in UART and Ethernet, respectively. From this, we concluded that where the nanoparticle size is larger than 430 atoms, Ethernet communication is preferable in comparison to UART and PCIe, and if the nanoparticle size is less than 430 atoms, then UART is more efficient. Both UART and Ethernet communication are robust, hot-pluggable, and user-friendly. This can lead to low-cost HPC for students and researchers who want to explore nanoparticles of experimental relevance. This application paves the way for the development of a Lab-on-a-Chip platform for the computation of IAP in the future.

4.4 Future Scope

UART and Ethernet communication-based systems were implemented for Au_{147} and Au_{309} nanoclusters, which are considered optimal sizes. The same UART and Ethernet communication-based MD simulation will be extended to larger clusters such as Au_{561}, Au_{571}, and Au_{923}.

Zynq platform FPGAs are advanced, sophisticated, and high-performance Xilinx FPGAs. Conventional FPGAs like Xilinx's Kintex-7, Artix-7, and Spartan series use softcore IPs like the Microblaze processor, AXI UartLite, and AXI EthernetLite. However, in the Zynq platform, these IPs are embedded on the FPGA chip as hardcore IPs, providing significantly higher performance. Therefore, the next goal is to implement MD simulation on Zynq-based FPGAs to further improve computation time. Additionally, the Zynq platform's MPSoC FPGAs have multiple cores and processors available on the FPGA chip. To reduce computation time, one possibility is to eliminate the role of the host computer and implement the complete system directly on multiple processors, thus removing the communication overhead between the host computer and the FPGA. These are the future extensions of this work.

Bibliography

[1] A. Boutros and V. Betz, "Fpga architecture: Principles and progression," *IEEE Circuits and Systems Magazine*, vol. 21, no. 2, pp. 4–29, 2021.

[2] M. Cummings and S. Haruyama, "Fpga in the software radio," *IEEE communications Magazine*, vol. 37, no. 2, pp. 108–112, 1999.

[3] J. Rettkowski, A. Boutros, and D. Göhringer, "Hw/sw co-design of the hog algorithm on a xilinx zynq soc," *Journal of Parallel and Distributed Computing*, vol. 109, pp. 50–62, 2017.

[4] A. Bitar, M. S. Abdelfattah, and V. Betz, "Bringing programmability to the data plane: Packet processing with a noc-enhanced fpga," in *2015 International Conference on Field Programmable Technology (FPT)*, IEEE, 2015, pp. 24–31.

[5] R. W. Hartenstein and H. Grünbacher, *Field-Programmable Logic and Applications. The Roadmap to Reconfigurable Computing: 10th International Conference, FPL 2000 Villach, Austria, August 27-30, 2000 Proceedings*. Springer Science & Business Media, 2000.

[6] A. Boutros, B. Grady, M. Abbas, and P. Chow, "Build fast, trade fast: Fpga-based high-frequency trading using high-level synthesis," in *2017 International Conference on ReConFigurable Computing and FPGAs (ReConFig)*, IEEE, 2017, pp. 1–6.

[7] T. Sterling, M. Brodowicz, and M. Anderson, *High performance computing: modern systems and practices*. Morgan Kaufmann, 2017.

[8] E. Strohmaier, J. J. Dongarra, H. W. Meuer, and H. D. Simon, "Recent trends in the marketplace of high performance computing," *Parallel Computing*, vol. 31, no. 3-4, pp. 261–273, 2005.

[9] J. J. Dongarra, "An overview of high performance computing and challenges for the future.," in *VECPAR*, 2008, p. 1.

[10] Y. Li, X. Zhao, and T. Cheng, "Heterogeneous computing platform based on cpu+ fpga and working modes," in *2016 12th International conference on computational intelligence and security (CIS)*, IEEE, 2016, pp. 669–672.

[11] M. Vestias and H. Neto, "Trends of cpu, gpu and fpga for high-performance computing," in *2014 24th International Conference on Field Programmable Logic and Applications (FPL)*, IEEE, 2014, pp. 1–6.

[12] R. K. Raj, C. J. Romanowski, J. Impagliazzo, *et al.*, "High performance computing education: Current challenges and future directions," in *Proceedings of the Working Group Reports on Innovation and Technology in Computer Science Education*, 2020, pp. 51–74.

[13] M. Kucharczyk and G. Dziwoki, "Simple communication with fpga device over ethernet interface," in *Computer Networks: 20th International Conference, CN 2013, Lwówek Śląski, Poland, June 17-21, 2013. Proceedings 20*, Springer, 2013, pp. 290–299.

[14] L. Shi, H. Chen, J. Sun, and K. Li, "Vcuda: Gpu-accelerated high-performance computing in virtual machines," *IEEE Transactions on computers*, vol. 61, no. 6, pp. 804–816, 2011.

[15] J. Fang, K. Zhou, M. Zhang, and W. Xiang, "Resource scheduling strategy for performance optimization based on heterogeneous cpu-gpu platform," *Computers, Materials & Continua*, vol. 73, pp. 1621–1635, 2022.

[16] M. C. Herbordt, T. VanCourt, Y. Gu, *et al.*, "Achieving high performance with fpga-based computing," *Computer*, vol. 40, no. 3, pp. 50–57, 2007.

[17] S. Kestur, J. D. Davis, and O. Williams, "Blas comparison on fpga, cpu and gpu," in *2010 IEEE computer society annual symposium on VLSI*, IEEE, 2010, pp. 288–293.

[18] S. S. Bulusu and S. Vasudevan, "Fpga accelerator for machine learning interatomic potential-based molecular dynamics of gold nanoparticles," *IEEE Access*, vol. 10, pp. 40 338–40 347, 2022.

[19] S. Jindal, S. Chiriki, and S. S. Bulusu, "Spherical harmonics based descriptor for neural network potentials: Structure and dynamics of au147 nanocluster," *The Journal of chemical physics*, vol. 146, no. 20, 2017.

[20] S. Páll, A. Zhmurov, P. Bauer, *et al.*, "Heterogeneous parallelization and acceleration of molecular dynamics simulations in gromacs," *The Journal of Chemical Physics*, vol. 153, no. 13, 2020.

[21] T. Fountain, A. McCarthy, F. Peng, *et al.*, "Pci express: An overview of pci express, cabled pci express and pxi express," in *10th ICALEPCS Int. Conf. on Accelerator & Large Expt. Physics Control Systems*, 2005.

[22] A. Boutros and V. Betz, "Fpga architecture: Principles and progression," *IEEE Circuits and Systems Magazine*, vol. 21, no. 2, pp. 4–29, 2021. DOI: 10.1109/MCAS.2021.3071607.

[23] P. Babu and E. Parthasarathy, "Reconfigurable fpga architectures: A survey and applications," *Journal of The Institution of Engineers (India): Series B*, vol. 102, pp. 143–156, 2021.

[24] *7 series FPGAs data sheet: Overview (DS180)*, Xilinx, 2020. [Online]. Available: https://docs.amd.com/v/u/en-US/ds180_7Series_Overview.

[25] *7 series FPGAs configurable logic block user guide (UG474)*, Xilinx, 2016. [Online]. Available: https://docs.amd.com/v/u/en-US/ug474_7Series_CLB.

[26] K. Tatas, K. Siozios, N. Vasiliadis, *et al.*, "Fpga architecture design and toolset for logic implementation," in *Integrated Circuit and System Design. Power and Timing Modeling, Optimization and Simulation: 13th International Workshop, PATMOS 2003, Turin, Italy, September 10-12, 2003. Proceedings 13*, Springer, 2003, pp. 607–616.

[27] *7 series DSP48e1 slice user guide (UG479)*, Xilinx, 2018. [Online]. Available: https://docs.amd.com/v/u/en-US/ug479_7Series_DSP48E1.

[28] *7 series FPGAs memory resources user guide (UG473)*, Xilinx, 2019. [Online]. Available: https://docs.amd.com/v/u/en-US/ug473_7Series_Memory_Resources.

[29] *7 series fpgas clocking resources user guide (UG472)*, Xilinx, 2018. [Online]. Available: https://docs.amd.com/v/u/en-US/ug472_7Series_Clocking.

[30] *Vivado design suite user guide : Design flows overview (UG892)*, Xilinx, 2022. [Online]. Available: https://www.xilinx.com/support/documents/sw_manuals/xilinx2022_1/ug892-vivado-design-flows-overview.pdf.

[31] *Vivado design suite user guide: System-level design entry (UG895)*, Xilinx, 2016. [Online]. Available: https://docs.amd.com/r/2021.1-English/ug895-vivado-system-level-design-entry/Revision-History.

[32] *Vivado design suite user guide: Designing with IP (UG896)*, Xilinx, 2021. [Online]. Available: https://docs.amd.com/r/2021.2-English/ug896-vivado-ip/Revision-History.

[33] *Vivado design suite tutorial: High-level synthesis (UG871)*, Xilinx, 2020. [Online]. Available: https://docs.amd.com/v/u/en-US/ug871-vivado-high-level-synthesis-tutorial.

[34] *Vivado design suite user guide: Logic simulation (UG900)*, Xilinx, 2021. [Online]. Available: https://docs.amd.com/r/2021.2-English/ug900-vivado-logic-simulation/Revision-History.

[35] *Vivado design suite user guide: Synthesis (UG901)*, Xilinx, 2021. [Online]. Available: https://docs.amd.com/v/u/2021.2-English/ug901-vivado-synthesis.

[36] *Vivado design suite user guide: Implementation (UG904)*, Xilinx, 2021. [Online]. Available: https://docs.amd.com/r/2021.2-English/ug904-vivado-implementation/Revision-History.

[37] *Vivado design suite user guide: Programming and debugging (UG908)*, Xilinx, 2021. [Online]. Available: https://docs.amd.com/r/2021.2-English/ug908-vivado-programming-debugging/Revision-History.

[38] *Kc705 evaluation board for the kintex-7 fpga user guide (UG810)*, Xilinx, 2019. [Online]. Available: https://docs.amd.com/v/u/en-US/ug810_KC705_Eval_Bd.

[39] E. Peña and M. G. Legaspi, "Uart: A hardware communication protocol understanding universal asynchronous receiver/transmitter," *Visit Analog*, vol. 54, no. 4, pp. 1–5, 2020.

[40] A. K. Gupta, A. Raman, N. Kumar, and R. Ranjan, "Design and implementation of high-speed universal asynchronous receiver and transmitter (uart)," in *2020 7th International Conference on Signal Processing and Integrated Networks (SPIN)*, IEEE, 2020, pp. 295–300.

[41] L. Cao, J. Chen, and J. Li, "Working principle and application analysis of uart," in *2023 IEEE 2nd International Conference on Electrical Engineering, Big Data and Algorithms (EEBDA)*, IEEE, 2023, pp. 255–259.

[42] S. Kumar, S. Dalal, and V. Dixit, "The osi model: Overview on the seven layers of computer networks," *International Journal of Computer Science and Information Technology Research*, vol. 2, no. 3, pp. 461–466, 2014.

[43] M. N. Sadiku and C. M. Akujuobi, "14.1 osi reference model," *Computers, Software Engineering, and Digital Devices*, 2018.

[44] R. Hollenbeck, "The ieee 802.3 standard (ethernet): An overview of the technology," 2001.

[45] J. Sommer, S. Gunreben, F. Feller, *et al.*, "Ethernet–a survey on its fields of application," *IEEE Communications Surveys & Tutorials*, vol. 12, no. 2, pp. 263–284, 2010.

[46] J.-M. Muller, N. Brisebarre, F. De Dinechin, *et al.*, *Handbook of floating-point arithmetic*. Springer, 2018.

[47] J. G. Tong, I. D. Anderson, and M. A. Khalid, "Soft-core processors for embedded systems," in *2006 International Conference on Microelectronics*, IEEE, 2006, pp. 170–173.

[48] *Microblaze processor reference guide (UG984)*, Xilinx, 2021. [Online]. Available: `https://www.xilinx.com/content/dam/xilinx/support/documents/sw_manuals/xilinx2021_2/ug984-vivado-microblaze-ref.pdf`.

[49] *Axi uart lite v2.0 product guide (PG142)*, Xilinx, 2017. [Online]. Available: https://docs.amd.com/v/u/en-US/pg142-axi-uartlite.

[50] *Axi ethernet lite mac logicore ip product guide (PG135)*, Xilinx, 2021. [Online]. Available: https://docs.amd.com/r/en-US/pg135-axi-ethernetlite/AXI-Ethernet-Lite-MAC-v3.0-LogiCORE-IP-Product-Guide.

[51] *7 series fpgas memory interface solutions user guide (UG586)*, Xilinx, 2012. [Online]. Available: https://docs.amd.com/v/u/1.4-English/ug586_7Series_MIS.

[52] *Axi interconnect logicore ip product guide (PG059)*, Xilinx, 2022. [Online]. Available: https://docs.amd.com/r/en-US/pg059-axi-interconnect/AXI-Interconnect-v2.1-LogiCORE-IP-Product-Guide.

[53] M. B. Gokhale and L. Shannon, "Fpga computing.," *IEEE Micro*, vol. 41, no. 4, pp. 6–7, 2021.

[54] *Vivado design suite user guide: Embedded processor hardware design (UG898)*, Xilinx, 2021. [Online]. Available: https://docs.amd.com/v/u/2021.1-English/ug898-vivado-embedded-design.

[55] *Ultrafast embedded design methodology guide (UG1046)*, Xilinx, 2018. [Online]. Available: https://docs.amd.com/v/u/en-US/ug1046-ultrafast-design-methodology-guide.