HIGH PERFORMANCE COMPUTING USING FPGA

MTech Thesis

By

Dharmendra Kartikey

DEPARTMENT OF ELECTRICAL ENGINEERING

INDIAN INSTITUTE OF TECHNOLOGY INDORE

May 2024

High Performance Computing using FPGA

A THESIS

Submitted in partial fulfilment of the requirements for the award of the degree
of
Master of Technology

by
Dharmendra Kartikey

DEPARTMENT OF ELECTRICAL ENGINEERING
INDIAN INSTITUTE OF TECHNOLOGY INDORE
May 2024

INDIAN INSTITUTE OF TECHNOLOGY

INDORE

CANDIDATE'S DECLARATION

I hereby certify that the work which is being presented in this thesis titled **High Performance computing using FPGA** in the partial fulfilment of requirement for award of the degree of **MASTER OF TECHNOLOGY** 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 June 2023 to May 2024 under the supervision of **Prof .Srivathsan Vasudevan, Department of Electrical engineering** and **Prof. Satya S Balusu ,Department of Chemistry** of 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.

Dartikey 24/05/24

Signature of the student with date
(DHARMENDRA KARTIKEY)

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

V Srivathsan 24/05/24

Signature of M.Tech supervisor with Date
(Prof Srivathsan Vasudevan)

B.S.Balusu 24.05.24

Signature of M.Tech supervisor with Date
(Prof Satya S Bulusu)

Dharmendra Kartikey has successfully given his M.Tech Oral Examination held on 7th May 2024.

V Srivathsan 24/05/24

Signature of M.Tech supervisor
Date
(Prof Srivathsan Vasudevan)

B.S.Balusu 24.05.24

Signature of M.Tech supervisor
Date
(Prof Satya S Bulusu)

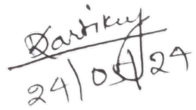
Convener,DPGC
Date

ACKNOWLEDGEMENTS

I would like to extend my deepest gratitude to **Prof. Dr. Srivathsan Vasudevan** and **Prof. Dr. Satya S Bulusu**, my thesis supervisors. Their unwavering support from the very beginning has been invaluable, and I am immensely grateful for the time they dedicated to guiding and motivating me, especially when faced with numerous options or during challenging times. I am thankful for the opportunities they provided, which have significantly shaped my MTech journey.

I would also like to express my appreciation to all the members and seniors of my lab for their technical guidance and support throughout my MTech thesis project. Additionally, I sincerely acknowledge the support of **IIT Indore** for providing the necessary lab equipment and facilities.

I am also grateful to **DRDO** for sponsoring my MTech program. Lastly, my heartfelt thanks go to my family, whose tremendous support and encouragement have been crucial in helping me stay positive and overcome various obstacles. To them, I will always be deeply grateful.



(DHARMENDRA KARTIKEY)

Dedicated to

My family

ABSTRACT

Molecular dynamics (MD) simulations involve calculating the forces Between atoms and the total energy of chemical system; however, these computations typically rely on high-end, sequential, and power-intensive servers. This poses a limitation when attempting to simulate large-scale systems that are relevant to real-world experiments. To address this challenge, An Artificial Neural Network (ANN) method to compute interatomic forces and energy in a system consisting of 147 Au atoms was created and it was implemented on a FPGA system.

Existing approach involves a combination of parallel computation on a Field Programmable Gate Array (FPGA) and simple computations performed on a Host PC. Effective communication between the Host PC and the FPGA is crucial for the success of this hardware- software co-design.

This Thesis is an exploratory thesis which tries to explore new methods of time performance improvements over existing approach. It investigates interrupt-based transmission and CDMA based data transfer method to improve time performance of existing design. Detailed comparative test cases have been devised to come to a quantitative conclusion.

At the end it proposes a Lab-on-a-chip architecture for Molecular dynamics cycle for enhancing time performance by eliminating inter-device communication like UART/Ethernet and making design more compact. It also covers basic design challenges in the Lab-on-a-chip architecture and provides solutions and test cases for validation of those solutions.

Contents

LIST OF FIGURES

LIST OF TABLES

CHAPTER 1

Introduction

1.1 Background

High Performance Computing (HPC) refers to the ability of a computer system to process data at a significantly faster speed. Typically, HPC systems are built by connecting multiple computer architectures, such as CPUs and GPUs, in a cluster. This allows multiple software programs to be run simultaneously on the cluster, resulting in faster processing times. HPC servers utilize various types of processors, including conventional CPUs and graphics processing units (GPUs). These components work together to improve computing power and speed, which are crucial for many HPC applications. GPUs, in particular, excel at processing large amounts of data in parallel, making them well-suited for high-performance computing tasks. The HPC server is bulky, power hungry and needs maintenance regularly. Hence there was a requirement find out the alternate to this to implement the Molecular dynamics cycles.

Field Programmable Gate Arrays (FPGAs) proved to be best alternate for this requirement. The FPGA supports high lever of parallelism which is ideal property for making hardware accelerators. They offer flexibility and have proven effective in enhancing computing power for certain applications.

The Molecular dynamics Cycle has been successfully implemented in our lab using FPGA and host PC by implementing Molecular dynamics calculation or inter atomic potential calculation on FPGA and verlet algorithm on Host PC and UART or ethernet as inter device communication. This design has time performance improvements over HPC server. It is compact, power saving and easy to maintenance.

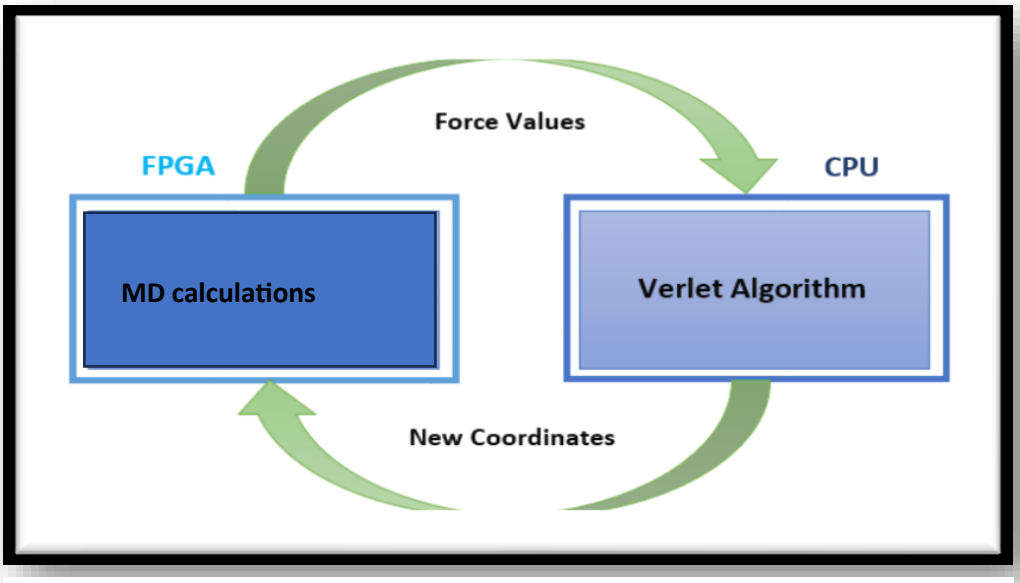


Figure 1. 1 Molecular Dynamics cycle implementation in FPGA

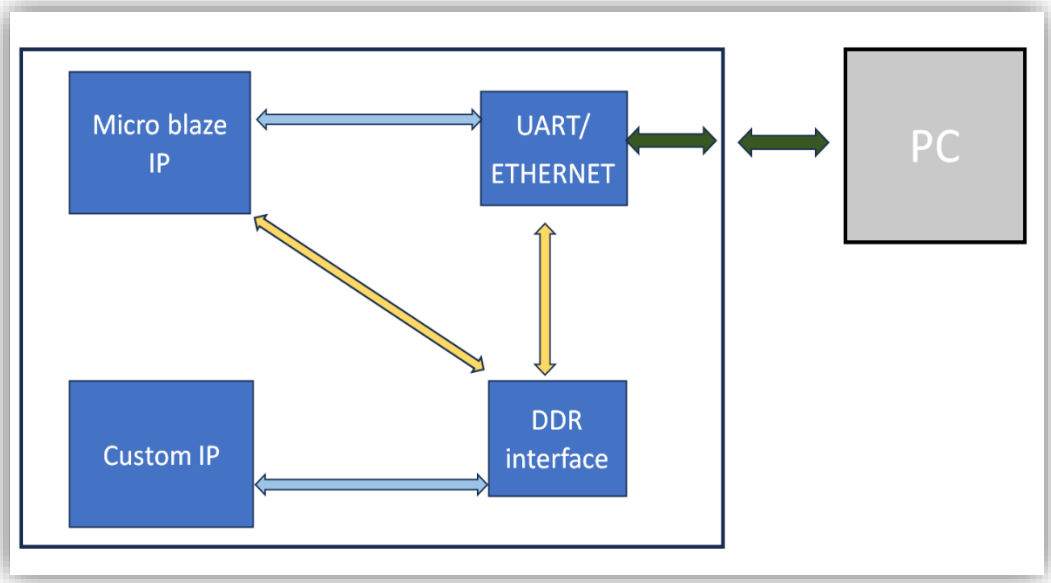


Figure 1. 2 Inter atomic potential/ MD calculations implementation in FPGA

Figure 2 shows IAP/MD calculations implemented inside FPGA.This block takes values of x,y,z coordinated from PC using UART/Ethernet .Microblaze sends the data to DDR then starts the Custom IP.This custom IP is already designed and validated IP .It takes coordinates from DDR and then saves back forces and energy values to some other memory location of DDR .Microblaze then send these data back to PC using same protocol .Verlet algorithm implemented in PC then takes force values and pro esses it and gives back coordinates values

1.2 Motivation

The primary focus of this project is to find out methods to improve time performance of existing design shown in figure 1 & figure 2. The existing design is distributed design where two functional module namely MD calculations and Verlet algorithm, are implemented in two different platforms . The basic motivation is to implement the two functional module into same platform so that the design can be more compact and the inter device communication can be avoided so as to save time consumed by inter device communication and hence making faster system.

1.3 Objective of This project

The objective of this project is to explore the methods or mechanism to provide time performance improvements over the existing design .It investigates following methods for achieving same objective

- UART interrupt based trans reception
- DMA data transfer between UART/Ethernet to DDR
- ZynqMPSOC based Lab -on-a-chip design

Making test design to compare these methods w.r.t existing design and result comparison and analysis are some of the objective

1.4 Organization of the Thesis

Chapter 1: Introduction, motivation and objectives

Chapter 2 UART polling and Interrupt based transmission and reception ,A comparative study and conclusions

Chapter 3 Microblaze and CDMA based data transfer between ethernet IP and BRAM ,A comparative study and conclusions

Chapter 4 Design based on ZynqMPSOC ,the embedded linux i.e petalinux ,its associated and framework for Lab-on-a chip

Chapter 5 Conclusion and Future Work

CHAPTER 2

UART Polling Vs Interrupt method

2.1 Literature survey

2.1.1 FPGA

FPGA is an integrated circuit that can be programmed after it has been manufactured. A FPGA is made up of a matrix of Configurable Logic Blocks (CLBs) which are interconnected through programmable interconnects. An FPGA consists of memory cells, control logic blocks, and interconnects. The word "field" in the name refers to the user's capacity to program the gate array at the field. The term "array" refers to a set of logic gates that can be programmed by end users in columns and rows.

During the time of FPGA configuration, the internal components are connected to each other via the interconnects in such a way that creates the hardware implementations for software applications FPGA devices deliver the high performance, low power, short time to market, reliability, high end productivity and flexibility in re-programmability in the hardware, because of their parallel nature. The flexibility provided by FPGA enhances performance by lowering system time complexity and allowing complicated logic to be implemented in real time. Our application requirements will determine which FPGA family is most suited for the job

2.1.1.1 Internal Architecture of an FPGA

Architecture of FPGA consists of three basic components;

1. Configurable Logic Blocks (CLBs) or Adaptive Logical Modules (ALM) which implement the logic functions. CLBs perform user- specified logical functionalities. Programmable interconnects, which provide the interconnection between CLBs or ALMs. Interconnect resources carry signals among blocks.

2. Programmable I/O blocks which creates an interface between internal array of logic blocks (CLBs) and devices external package pins.

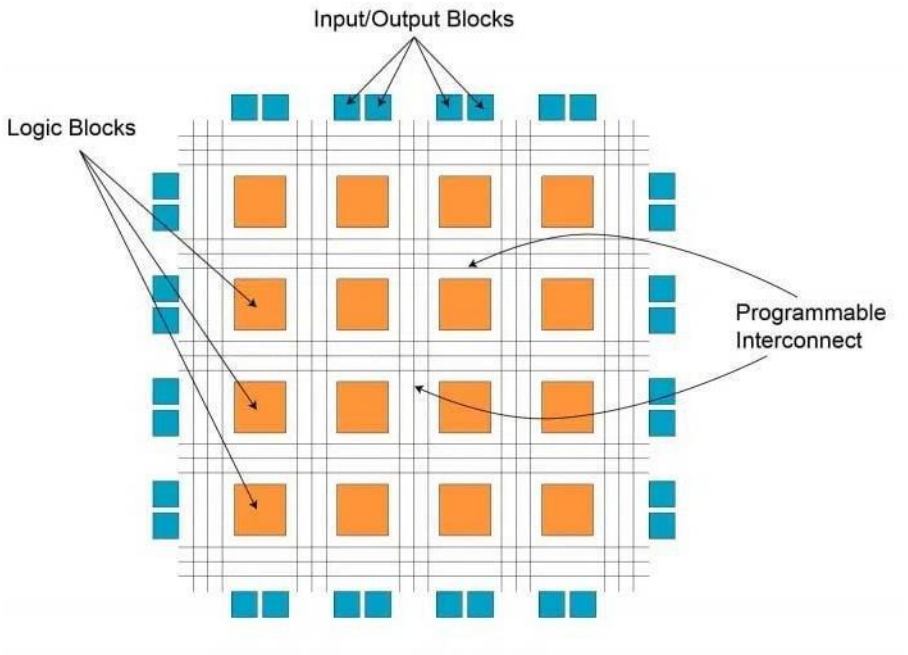


Figure 2. 1 Internal Architecture of FPGA

2.1.1.2 Configurable Logic Block (CLB)

CLBs are the functional elements that allow the user to develop logic using the basic building blocks. Lookup Tables (LUTs) and DSP blocks (for multiplication and subtraction) are used to create arbitrary logic functions. Flip-Flops (FF) for clocked storage elements are also included. CLB also has signal routing elements that allow signals to be routed from one block to another. They give physical support for a design that has been implemented and downloaded. CLBs have inputs on both sides, making them versatile in terms of logic mapping and division. The design given below shows the implementation of a 4 input CLB using 2 3 input LUTs, a full adder, a D-flipflop and 2 multiplexers.

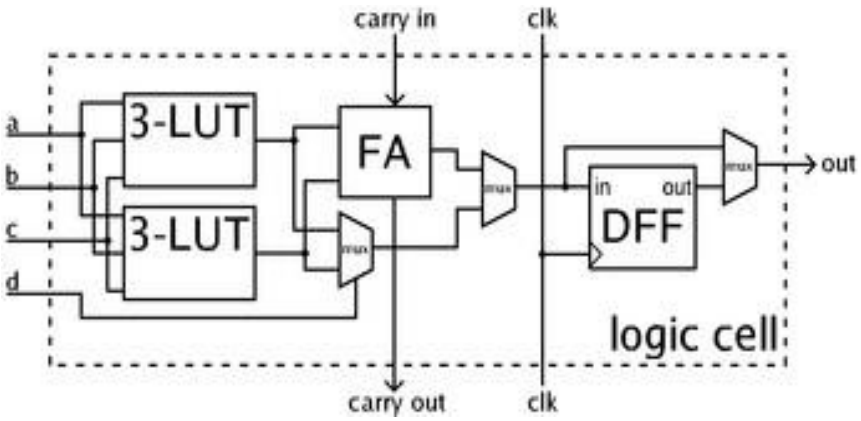


Figure 2. 2 Configurable Logic Block

2.1.1.3 Programmable Interconnects

The device's programmable interconnect is a massive programmable switch matrix that allows signals from all portions of the device to reach all other parts of the device. The programmable interconnects are implemented using 3 technologies-

- SRAM
- Flash Memory
- Antifuse

2.1.1.4 Programmable IOB

A programmable I/O block is used to bring signals from the outside world onto the device and transfer them back. With tri-state and open collector output control, it comprises of an input buffer and an output buffer. On output terminals, there are usually pull-up resistors and sometimes pull-down resistors. In addition, many outputs have an FF that may be used to provide a registered output to external boards or instruments. So naturally these outputs only change when a clock edge enables them to transit.

2.1.2 Microblaze IP

Microblaze is a Xilinx soft core IP that implements a microprocessor entirely within the FPGA's general-purpose memory and logic fabric. The MicroBlaze has an excellent connection architecture that allows it to serve a wide range of embedded applications. The AXI connection, MicroBlaze's primary I/O bus, is a system-memory mapped transaction bus with master–slave functionality. The Core Connect PLB bus was used in previous MicroBlaze versions. AXI is directly interfaced by the majority of FPGA vendor-supplied and third-party IP (or through an AXI interconnect). MicroBlaze employed a dedicated LMB bus, which provides fast on-chip storage, to access local memory (FPGA RAM). AXI4-Stream connections are used to facilitate user-defined coprocessors. By outsourcing sections or the entire calculation to a user- designed hardware module, the coprocessor(s) interface can speed up computationally complex computations.

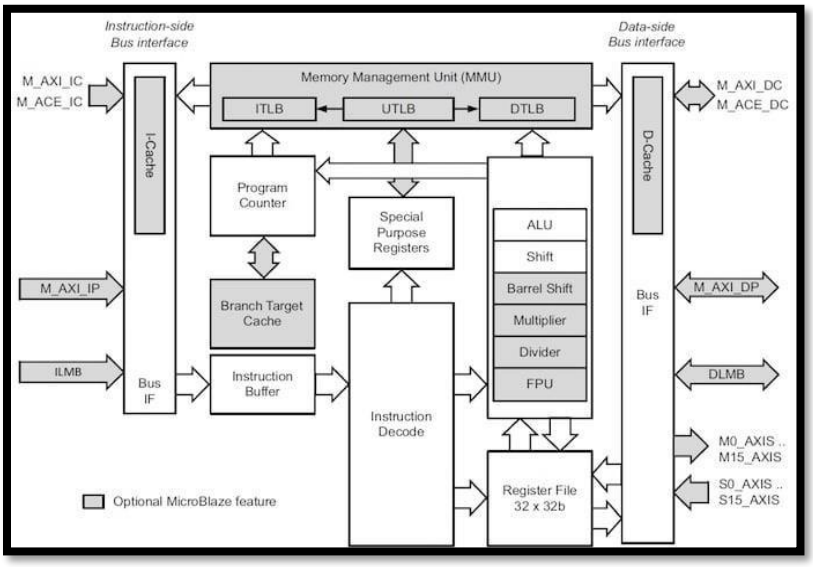


Figure 2. 3 Microblaze Internal Architecture

Features

- 32 32-bit general Purpose registers
- 32-bit instruction word in three operands and two addressing modes (Immediate and Indexed)
- Use 32-bit address bus
- It uses single issue pipeline. (3 stage/5 stage/ 8 stage)

2.1.3 UARTlite IP

The AXI Universal Asynchronous Receiver Transmitter (UART) Lite core serves as the I/O device of the system. This IP uses UART communication protocol for off-chip communication. UART signals are interfaced to the AXI interface of the Advanced Microcontroller Bus Architecture (AMBA), and a transmit data FIFO and receive data FIFOis present between them to synchronize between the high-speed on-chipcommunication and low speed off-chip communication. The AXI4-Lite protocol is being used to interface with the control register of the IP'

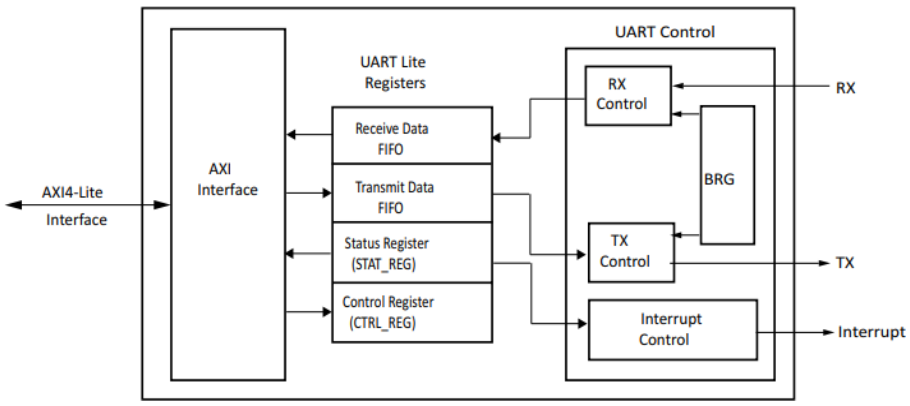


Figure 2. 4 AXI UARTLite Internal Architecture

AXI Interface: This is used to implement the AXI4-Lite slave interface, which allows access to registers and data transfer

between modules.

- UART Lite Registers: Memory mapped registers are included in this module. It consists of a control register, a status register, and a pair of 16-character transmit/receive FIFOs.

UART Control: This block consists of

1. Receiver Control: This block samples received data and writes it to the Receive Data FIFO based on the generated baud rate.

2. Transmitter Control: This block reads data from Transmit Data FIFO and sends it out on the UART transmitter interface.

3. Baud Rate Generator: This block generates various baud rates as per the User requirement.

4. Interrupt Control: The AXI UART Lite core provides interrupt enable/disable control. If interrupts are enabled, a rising-edge sensitive interrupt is generated when the receive FIFO becomes non-empty or when the transmit FIFO becomes empty.

2.2 UART transmission and reception process

2.2.1 Polling based Transmission process

Figure shows polling based transmission process .The register in UARlite IP provides TX empty flag .The data is written over UARTlite FIFO as long is TX empty flag is set .Once TX empty flag is reset .The micro blaze stops for writing and keep polling for this flag.As soon as it becomes set it again starts writing over FIFO. Since micro blaze is always busy checking the status of this TX empty flag ,it is called polling based Transmission process.

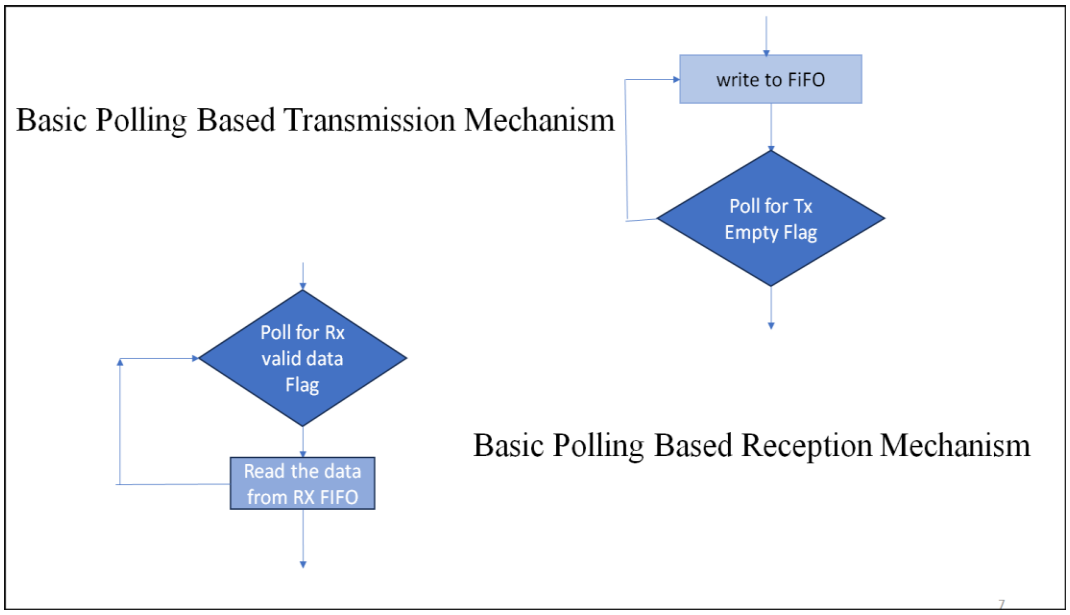


Figure 2. 5 Polling based Transmission and reception process

2.2.2 Polling based Reception process

Figure shows polling based reception process.The Microblaze keeps pooling for RX valid data flag .This data flag sets when atleast one data is available in RX FIFO.as soon as This data flag is set .The microblaze reads from the RX FIFO .as microblaze is busy polling for RX Valid data flag .Its caleed polling based reception.

2.2.3 Polling based Tx and Rx using high level function

The high-level functions provided by Device IP BSP for sending and receiving are XUartLite_Send and XUartLite_Recv. These functions take pointer to data structure XUartLite and sending data memory and number of bytes to be sent, as arguments. XUartLite structure has one of the element structure XUartLite_Buffer which has 3 elements *NextBytePtr,RequestedBytes and RemainingBytes,when this function are called first mapping process happens then internally XUartLite_SendBuffer/ XUartLite_RecvBuffer is called which works as figure .

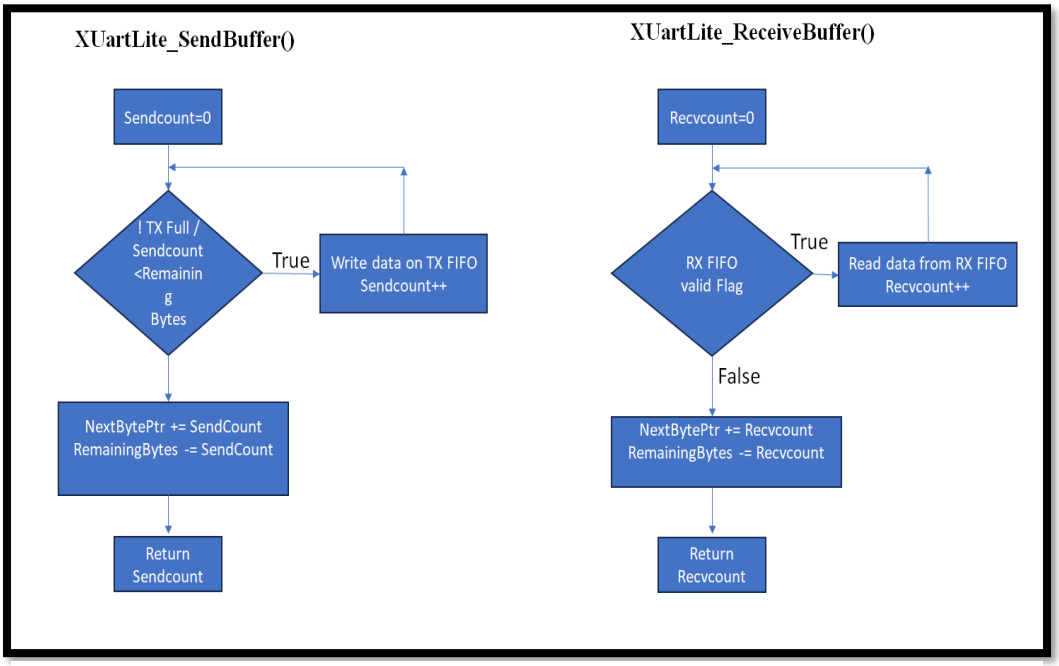


Figure 2. 6 UART polling based TX and RX using high level functions

2.2.4 Interrupt based Transmission process

Figure shows generic block design with UART and microblaze with interrupt controller. Interrupt is generated when “The RX FIFO becomes non empty or the TX FIFO becomes empty”. Whenever interrupt is generated the interrupt controller passes this interrupt to microblaze which call Interrupt Service Routine (ISR). Figure shows the flow when the XUartLite_Send function is called. First the FIFO becomes full by writing 16 bytes of data, the transmission process starts once transmission of 16 byte is over and TX FIFO becomes empty,hence interrupt is generated .This interrupt calls ISR (SendDataHandler) which again write another 16 bytes of data in FIFO,and this process continues. The ISR takes record of the sent count ,as soos as this becomes equal to desired data to be sent, the program stops.

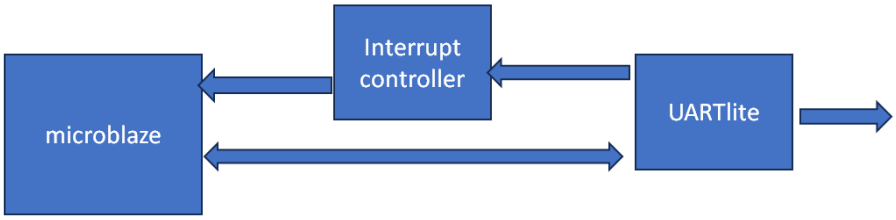


Figure 2. 7 UART interrupt based generic design

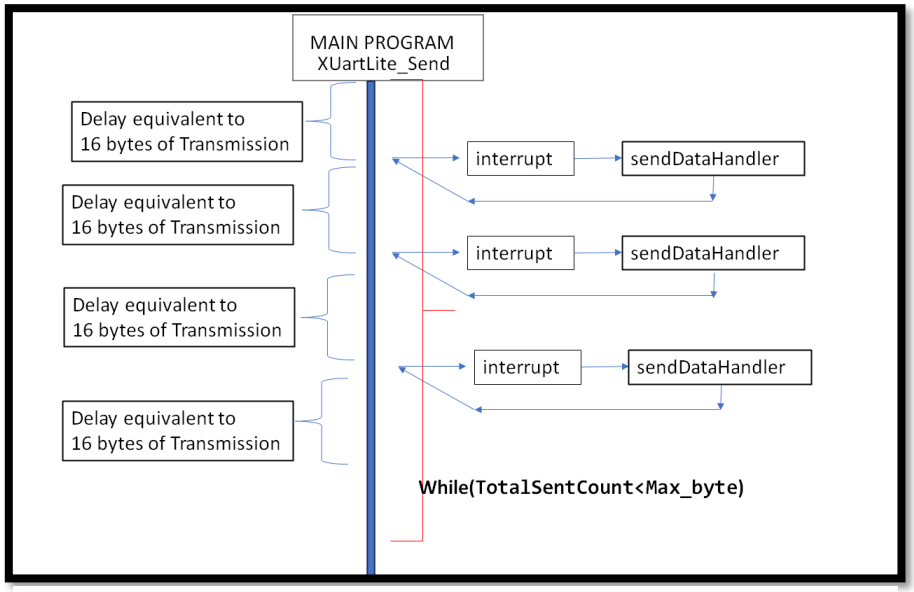


Figure 2. 8 UART interrupt based Transmission process

2.2.5 Interrupt based Reception process

Similar to transmission process, reception process also works, but the condition for interrupt generation is different here. Figure shows the reception process. Every time a byte is received, the RX FIFO becomes non empty and hence interrupt is generated. As interrupt is generated ,ISR (RecvDataHandler) is called then this FIFO data is read by microblaze.when Next byte is received again interrupt is generated and same process continues. The ISR takes record of the received count, as soon as this becomes equal to desired data to be received, the program stops.

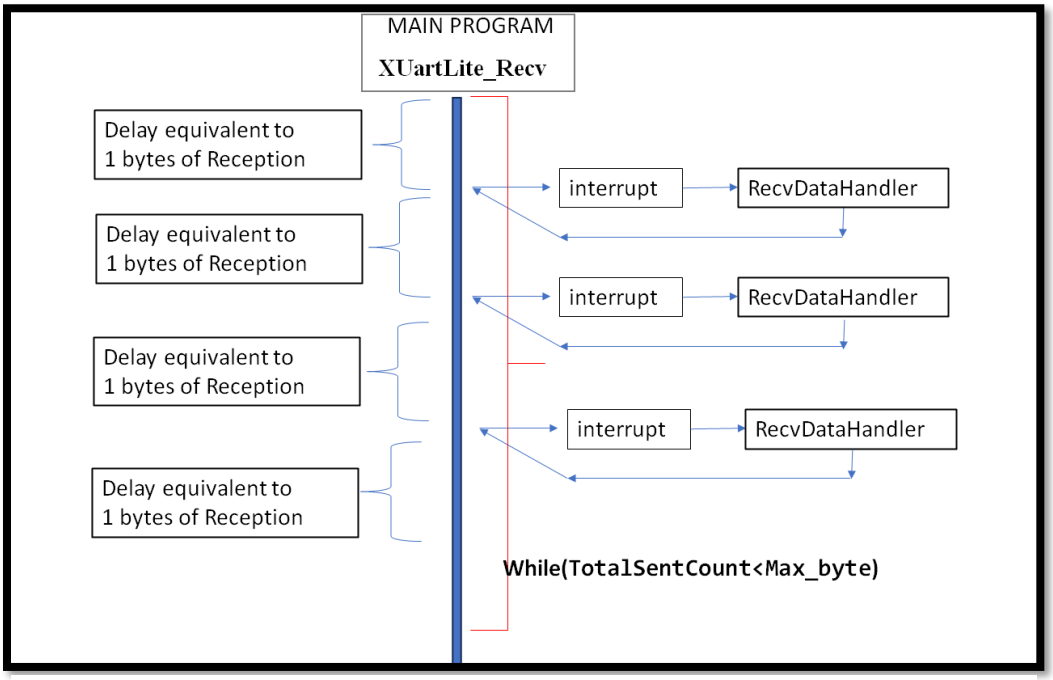


Figure 2. 9 UART interrupt-based Reception process

2.3 The comparison test case: UART Polling Vs Interrupt

2.3.1 Test objective

To measure the time duration of sending 1768 bytes of data from UART based polling and interrupt-based method and comparison of time performance.

2.3.2 Test specification

Test Board: Genesys 2

Vivado 2017.4

SDK 2017.4

UART baud rate =9600 bps

2.3.3 Test design

There were 2 test design made. Test design 1 was used for polling-based transmission method whereas test design 2 was used for interrupt-based transmission method. the AXI timer IP was used for time measurement.

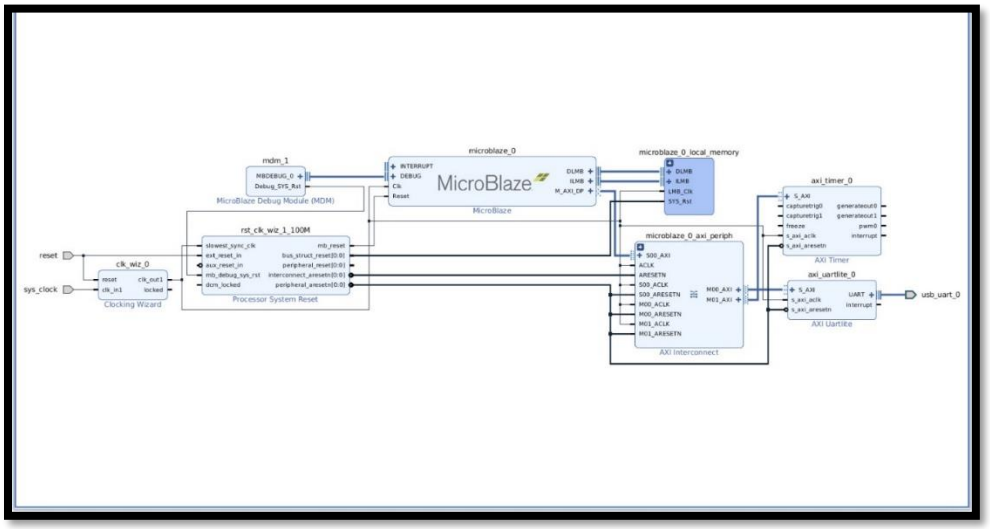


Figure 2. 10 Block design for test design 1 (polling-based Transmission)

```
#include <stdio.h>
#include "platform.h"
#include "xil_printf.h"
#include<xuartlite.h>
#include <xparameters.h>
#include<xuartlite_l.h>
#include <xtmrctr.h>


#define max_byte 50



int main()
{
    init_platform();
    u8 sendbuffer[max_byte];
    int sendcount=0;
    XUartLite a;
    XTmrCtr x;
    u32 time1=0,time2=0;


    XTmrCtr_Initialize(&x,XPAR_AXI_TIMER_0_DEVICE_ID);
    XUartLite_Initialize(&a,XPAR_AXI_UARTLITE_0_DEVICE_ID);
    XUartLite_ResetFifos(&a);

    /*
       * Enable the Autoreload mode of the timer counters.
       */
      XTmrCtr_SetOptions(&x, 0 , XTC_AUTO_RELOAD_OPTION);


    XTmrCtr_Start(&x, 0);


  time1 = XTmrCtr_GetValue(&x, 0);
  for(int i=0;i<16;i++)
           { sendbuffer[i] =53;

           }
           for(int i=16;i<32;i++)
           { sendbuffer[i] =54;

           }
           for(int i=32;i<48;i++)
           { sendbuffer[i] =55;
           }
           for(int i=48;i<max_byte;i++)
           { sendbuffer[i] =56;
           }
```

```
   //Transmitter module
    while(1)
        { sendcount+= XUartLite_Send(&a,sendbuffer+sendcount,max_byte-
sendcount);

         if (sendcount==max_byte)

            break;

        }

    time2 =XTmrCtr_GetValue(&x, 0);
    XTmrCtr_Stop(&x, 0);


    xil_printf("Toal delay in sending 50 numbers =%lu", time2-time1);

    cleanup_platform();
    return 0;
}
```

Figure 2. 11 C language code for test design 1 (polling-based Transmission)

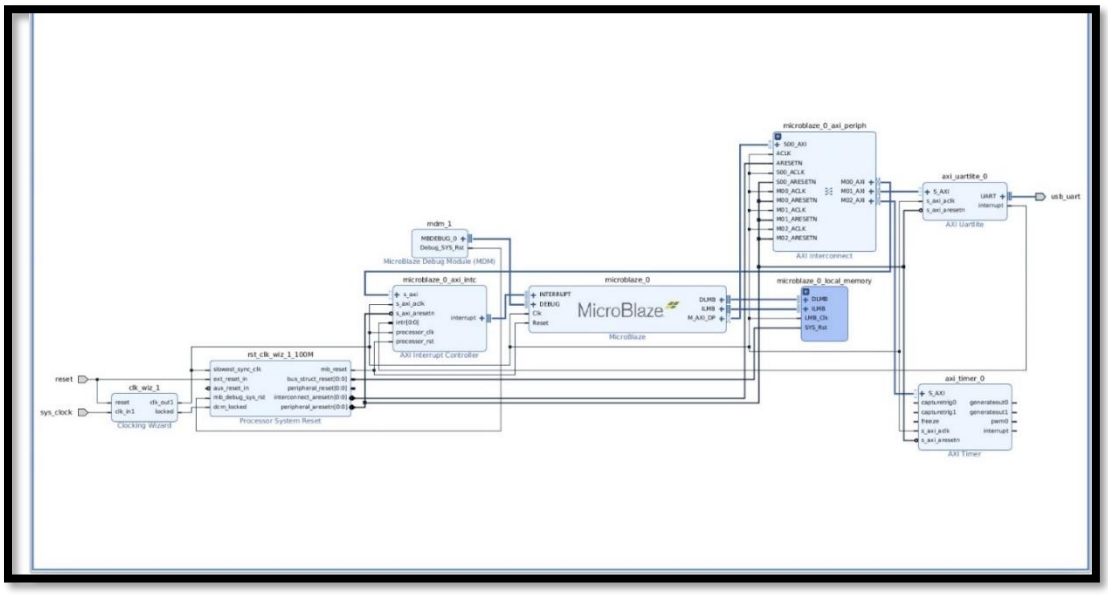


Figure 2. 12 Block design for test design 2 (interrupt-based Transmission)

```
#include "platform.h"
#include<stdio.h>
#include<xuartlite.h>
#include<xil_io.h>
#include<xil_printf.h>
#include<xuartlite_l.h>
#include<xintc.h>
#include<xil_exception.h>
#include<xparameters.h>
#include<sleep.h>
#include<xtmrctr.h>
#define MAX_BYTE 50

volatile static int TotalSentCount;


u8 sendbuffer[MAX_BYTE];
//u32 buffer[1000];
void SendHandler(void *CallBackRef, unsigned int EventData)
{


      TotalSentCount= EventData;
      //xil_printf(" total send data is =%d\r\n",TotalSentCount);

}

int main()
{     init_platform();
        XTmrCtr x;
      u32= time1,time2;
      XUartLite myuart;
      //XUartLite_Config *myuartconfig;
      XIntc InterruptController;
      //u32 rx_interrupt,tx_interrupt;
      for(int i=0;i<16;i++)
            { sendbuffer[i] =53;

            }
            for(int i=16;i<32;i++)
            { sendbuffer[i] =54;

            }
            for(int i=32;i<48;i++)
            { sendbuffer[i] =55;
            }
            for(int i=48;i<MAX_BYTE;i++)
            { sendbuffer[i] =56;
            }

        XTmrCtr_Initialize(&x,XPAR_AXI_TIMER_0_DEVICE_ID);
        XTmrCtr_SetOptions(&x,0,XTC_AUTO_RELOAD_OPTION);
        XTmrCtr_Start(&x,0);
```

```
      XUartLite_Initialize(&myuart,XPAR_AXI_UARTLITE_0_DEVICE_ID);
      //Initializing Uartlite ip
//    myuartconfig=XUartLite_LookupConfig(XPAR_AXI_UARTLITE_0_DEVICE_ID);
//
      XUartLite_CfgInitialize(&myuart,myuartconfig,XPAR_AXI_UARTLITE_0_BA
SEADDR);
      XUartLite_InterruptHandler(&myuart);

      //Configuring Interrupt Controller
      XIntc_Initialize(&InterruptController,XPAR_MICROBLAZE_0_AXI_INTC_DE
VICE_ID);
      XIntc_Connect(&InterruptController,
XPAR_INTC_SINGLE_DEVICE_ID,(XInterruptHandler)XUartLite_InterruptHandler,
(void*)&myuart);
      XIntc_Start(&InterruptController,XIN_REAL_MODE);
      XIntc_Enable(&InterruptController,XPAR_INTC_SINGLE_DEVICE_ID);

      //Configuring Exception register handler
      Xil_ExceptionInit();
      Xil_ExceptionRegisterHandler(XIL_EXCEPTION_ID_INT,(Xil_ExceptionHan
dler)XIntc_InterruptHandler,&InterruptController);
      Xil_ExceptionEnable();

      //Setting sendhandler
      XUartLite_SetSendHandler(&myuart,SendHandler,&myuart);



      XUartLite_EnableInterrupt(&myuart);//Enabling the interrupt

         time1=XTmrCtr_GetValue(&x,0);
      XUartLite_Send(&myuart,(u8*)sendbuffer,MAX_BYTE);
   // xil_printf("\n before loop \r\n");
    while(TotalSentCount<MAX_BYTE)
    {
    }
    time2=XTmrCtr_GetValue(&x,0);
    //xil_printf("after loop\r\n");

    xil_printf("Total delay in sending 50 numbers =%lu",(time2-time1));

    XIntc_Disconnect(&InterruptController, XPAR_INTC_SINGLE_DEVICE_ID);
    cleanup_platform();
    return 0;
}
```

Figure 2. 13 C language code for test design 2 (interrupt-based Transmission)

2.3.4 Test Results

Table 1: Test Results for Interrupt Vs Polling

Time	Estimated	Polling	interrupt
Transmission time for 50 bytes	0.05208333	0.05207787	0.05206633

2.3.5 Conclusion

It can be seen from results that the time for sending 50 bytes for polling and interrupt based method is same, however interrupt based method provide facility for doing additional activity during the sending time.

CHAPTER 3

Microblaze and DMA data transfer

3.1 Literature survey

3.1.1 Dynamic Memory Access (DMA)

Generic DMA, often referred to simply as DMA, is a fundamental component in computer architecture that enables efficient data transfers between peripheral devices and memory without involving the CPU. It is an essential feature in modern computing systems, providing improved performance by offloading data movement tasks from the CPU.

Key Features:

1. Data Transfer Efficiency: DMA controllers enhance system efficiency by allowing data to be transferred directly between peripherals and memory, bypassing the CPU. This reduces CPU overhead and improves overall system performance.

2. synchronous Operation: DMA operates asynchronously with the CPU, enabling concurrent data transfers while the CPU executes other tasks. This asynchronous operation enhances system throughput and responsiveness.

3. Block Transfer Support: DMA controllers support block data transfers, allowing large chunks of data to be moved efficiently between memory and peripherals. This is particularly beneficial for applications

requiring high-bandwidth data movement, such as multimedia processing and networking.

4. Programmable Configuration: DMA controllers often feature programmable configuration options, including transfer size, transfer direction, and addressing modes. This flexibility allows developers to tailor DMA operations to specific application requirements.

5. Interrupt Handling: DMA controllers typically support interrupt mechanisms to notify the CPU upon completion of data transfers or when specific events occur. This enables efficient handling of data transfer events and synchronization with CPU tasks.

Applications:

1. Storage Devices: DMA is commonly used in storage devices such as hard disk drives (HDDs), solid-state drives (SSDs), and optical drives to facilitate fast data transfers between storage media and system memory.

2. Networking: In networking applications, DMA accelerates data movement between network interfaces and system memory, enabling high-speed data transmission and processing in networking devices.

3. Graphics Processing: DMA plays a crucial role in graphics processing units (GPUs) and graphics cards, facilitating rapid transfer of image and video data between the GPU memory and system memory for rendering and display.

4. Multimedia Processing: DMA is essential for multimedia applications such as audio and video processing, enabling efficient transfer of multimedia data between peripherals and memory for playback, recording, and editing.

5. Embedded Systems: In embedded systems and microcontroller-based designs, DMA is used to optimize data transfer between peripherals and memory, conserving CPU resources and reducing power consumption.

Generic DMA is a vital component in modern computing systems, enabling efficient and high-speed data transfers between peripherals and memory. Its asynchronous operation, block transfer support, and programmable configuration options make it indispensable for a wide range of applications, including storage devices, networking, graphics processing, multimedia processing, and embedded systems. By offloading data movement tasks from the CPU, DMA enhances system performance, throughput, and responsiveness, contributing to overall system efficiency and functionality.

AXI CDMA IP: Optimizing Memory-Mapped Data Transfers in FPGA-Based Systems

Introduction:

Efficient data movement lies at the heart of optimized performance in FPGA-based systems. The Advanced eXtensible Interface (AXI) Central Direct Memory Access (CDMA) Intellectual Property (IP) emerges as a cornerstone solution, meticulously designed to expedite transfers between memory-mapped sources and destinations within the FPGA architecture. Its focused functionality and tailored approach to memory-to-memory operations make it an indispensable asset in FPGA development, empowering engineers to optimize system efficiency and data handling capabilities.

Key Features:

1. Memory-to-Memory Transfers:

• Unlike traditional DMA controllers, the AXI CDMA IP specializes in facilitating data transfers between memory-mapped sources and destinations within the FPGA architecture.

• This targeted approach streamlines data movement between distinct memory regions, minimizing latency and optimizing system-level data management.

2. Efficient Scatter-Gather Support:

• The AXI CDMA IP boasts robust scatter-gather capabilities, enabling non-contiguous data transfers between memory-mapped regions.

• This feature enhances flexibility and efficiency, particularly in applications with complex data transfer patterns or fragmented data structures.

3. Configurable Transfer Modes:

• Flexibility lies at the core of the AXI CDMA IP, offering configurable transfer modes tailored to diverse application demands.

• Burst transfers and streaming transfers empower developers to fine-tune data movement parameters, maximizing throughput and minimizing latency.

Applications:

1. Memory Management Optimization:

• Within FPGA-based systems, the AXI CDMA IP plays a pivotal role in memory management tasks, including data copying, memory initialization, and synchronization.

• It enables efficient utilization of memory resources, simplifying data handling complexities across multifaceted applications.

2. Data Processing Pipelines:

• Unveiling its prowess in data processing pipelines, the AXI CDMA IP fosters seamless data flow between disparate processing stages and memory buffers.

• This seamless integration minimizes latency, facilitating real-time processing capabilities in domains such as digital signal processing (DSP) and image processing.

3. High-Performance Computing (HPC):

• Anchoring high-performance computing endeavors, the AXI CDMA IP propels data movement efficiency to unprecedented heights.

• It accelerates data transfers between memory regions, processing units, and external storage devices, underpinning peak system performance and scalability.

In summary, the AXI CDMA IP emerges as a pivotal solution for optimizing memory-mapped data transfers within FPGA architectures. Its specialized functionality, robust scatter-gather support, and configurable transfer modes empower engineers to streamline data movement processes, enhancing system efficiency and responsiveness across a diverse range of applications.

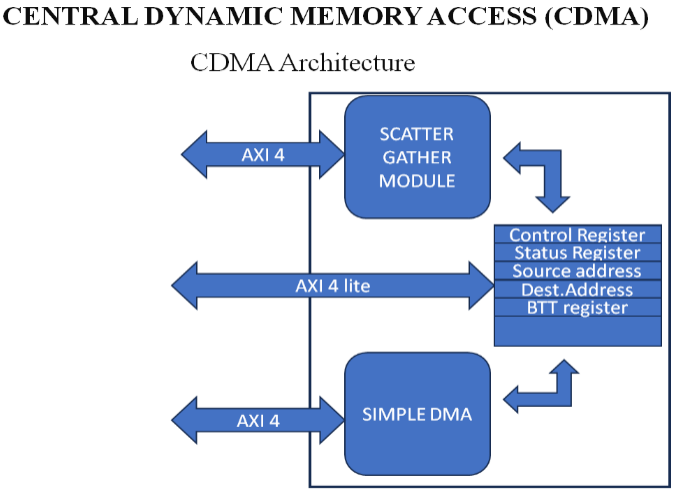
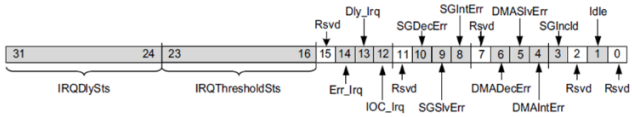


Figure 3. 1 AXI CDMA Architecture

SIMPLE DMA TRANSFER Sequence

- Verify that idle bit of the Status register is 1.



- Write source address to Source address (SA) register
- Write Destination address to Destination address (DA) register
- Initiate transfer by writing the number of bytes to the CDMA Bytes to Transfer (BTT) register

High level function **XAxiCdma_SimpleTransfer()**

Figure 3. 2 CDMA simple DMA Transfer sequence

AXI DMA IP: Streamlining Data Transfers in FPGA-Based Systems

Introduction:

In the realm of FPGA-based systems, efficient data movement is pivotal for optimizing overall performance. The Advanced eXtensible Interface (AXI) Direct Memory Access (DMA) Intellectual Property (IP) stands as a cornerstone solution, meticulously engineered to orchestrate seamless transfers between streaming sources or destinations and memory-mapped locations. Its versatile functionality and optimized design make it an indispensable asset in FPGA development, empowering engineers to enhance system throughput and responsiveness.

Key Features:

1. Stream-to-Memory Transfers:

• The AXI DMA IP excels in facilitating data transfers between streaming sources, such as peripherals or processing units, and memory-mapped destinations within the FPGA architecture.

- This capability streamlines the flow of data from streaming sources to designated memory regions, minimizing CPU intervention and optimizing system efficiency.

2. Efficient Data Movement:

- By offloading data transfer tasks from the CPU, the AXI DMA IP reduces processing overhead, enabling the CPU to focus on critical computational tasks.

- Its optimized data transfer mechanisms ensure high-bandwidth, low-latency communication, ideal for applications requiring rapid and continuous data streaming.

3. Configurable Transfer Modes:

The AXI DMA IP offers a range of configurable transfer modes, including burst and streaming modes, allowing developers to tailor data transfer parameters to specific application requirements.

This flexibility empowers engineers to optimize data throughput and latency, achieving optimal performance across diverse use cases.

Applications:

1. High-Performance Computing (HPC):

- In HPC applications, the AXI DMA IP accelerates data movement between processing units and memory, facilitating efficient parallel computation and maximizing system throughput.

2. Multimedia Processing:

- In multimedia processing tasks such as video rendering or audio processing, the AXI DMA IP enables seamless data streaming between memory and processing units, ensuring smooth and uninterrupted playback.

3. Networking:

• Within networking applications, the AXI DMA IP supports fast and reliable data transfers between network interfaces and memory, enabling efficient packet processing and network communication.

In conclusion, the AXI DMA IP serves as a foundational component in FPGA-based systems, streamlining data transfers between streaming sources and memory-mapped destinations. Its versatile functionality, efficient data movement mechanisms, and configurable transfer modes make it an invaluable asset across a diverse range of applications, from high-performance computing to multimedia processing and networking.

3.1.2 Ethernetlite IP

AXI4 Interface Module This module provides the interface to the AXI4 and implements AXI4 protocol logic. The AXI4 interface module is a bidirectional interface between the AXI Ethernet Lite MAC core and the AXI4/AXI4-Lite interface standard.

TX Buffer The TX Buffer module consists of 2K byte dual port memory to hold transmit data for one complete frame and the transmit interface control registers. It also includes optional 2K byte dual port memory for the pong buffer based on the parameter C_TX_PING_PONG.

AXI4 Interface Module:

This module provides the interface to the AXI4 and implements AXI4 protocol logic. The AXI4 interface module is a bidirectional interface between the AXI Ethernet Lite MAC core and the AXI4/AXI4-Lite interface standard

TX Buffer

The TX Buffer module consists of 2K byte dual port memory to hold transmit data for one complete frame and the transmit interface control

registers. It also includes optional 2K byte dual port memory for the pong buffer based on the parameter C_TX_PING_PONG

RX Buffer

The RX Buffer module consists of 2K dual port memory to hold receive data for one complete frame and the receive interface control register. It also includes optional 2K dual port memory for the pong buffer based on the parameter C_RX_PING_PONG

Transmit

This module consists of transmit logic, Cyclic Redundancy Check (CRC) generator module, transmit data mux, TX First In First Out (FIFO) and the transmit interface module. The CRC generator module calculates the CRC for the frame to be transmitted. The transmit control mux arranges this frame and sends the preamble, Start of Frame Delimiter (SFD), frame data, padding and CRC to the transmit FIFO in the required order. When the frame is transmitted to the PHY, this module generates a transmit interrupt and updates the transmit control register

Receive

This module consists of the RX interface, loopback control mux, RX FIFO, CRC checker and Receive Control module.

Receive data signals from the PHY are passed through the loopback control mux and stored in the RX FIFO. If

loopback is enabled, data on the TX lines is passed to the RX FIFO. The CRC checker module calculates the CRC of

the received frame and if the correct CRC is found, receive control logic generates the frame receive interrupt.

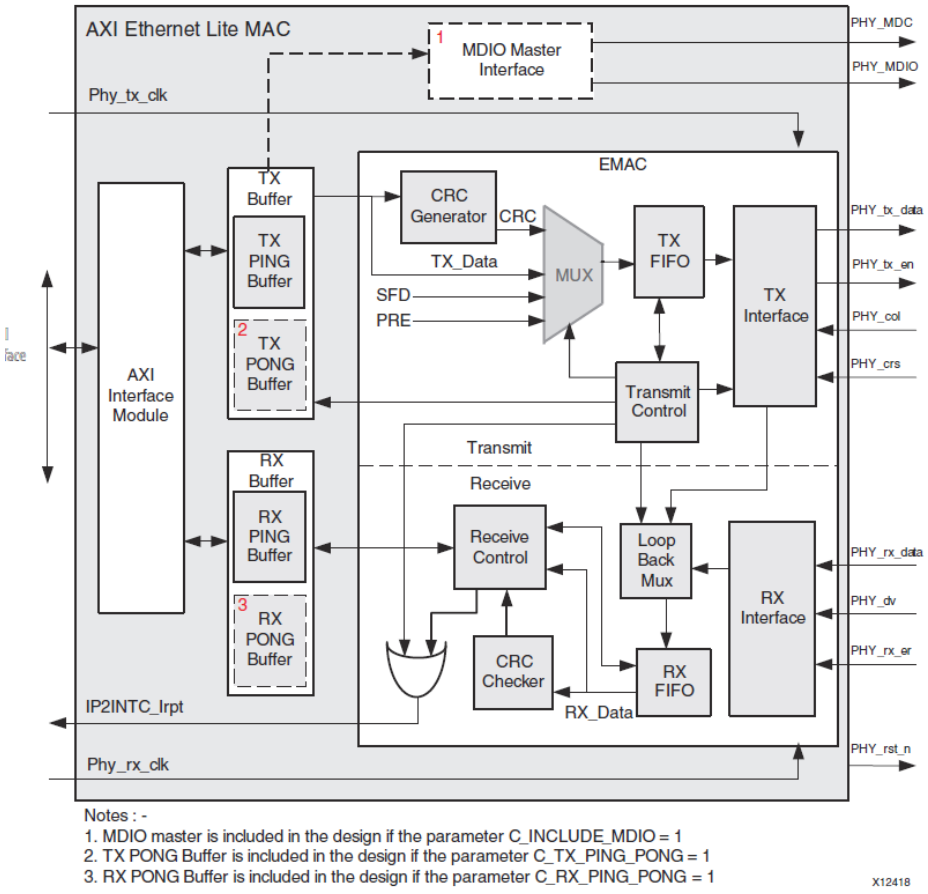


Figure 3. 3 AXI EthernetLite IP Block Diagram

Ethernet protocol:

Ethernet data is encapsulated. The fields and bits in the frame are transmitted from left to right (from the least significant bit to the most significant bit), unless specified otherwise.

Preamble

The preamble field is used for synchronization and must contain seven bytes with the pattern 10101010. If a collision is detected during the transmission of the preamble or start of frame delimiter fields, the transmission of both fields is completed.

For transmission, this field is always automatically inserted by the AXI Ethernet Lite MAC core and should never appear in the packet data provided to the AXI Ethernet Lite MAC core. For reception, this field is always stripped from the packet data. The AXI Ethernet Lite MAC design does not support the Ethernet 8-byte preamble frame type.

Start Frame Delimiter

The start frame delimiter field marks the start of the frame and must contain the pattern 10101011. If a collision is detected during the transmission of the preamble or start of frame delimiter fields, the transmission of both fields is completed.

The receive data valid signal from the PHY (PHY_dv) can go active during the preamble but is active prior to the start frame delimiter field. For transmission, this field is always automatically inserted by the AXI Ethernet Lite MAC core and should never appear in the packet data provided to the AXI Ethernet Lite MAC core. For reception, this field is always stripped from the packet data.

Destination Address

The destination address field is 6 bytes in length. The least significant bit of the destination address is used to determine if the address is an individual/unicast (0) or group/multicast (1) address. Multicast addresses are used to group logically related stations.

The broadcast address (destination address field is all 1's) is a multicast address that addresses all stations on the LAN. The AXI Ethernet Lite MAC supports transmission and reception of unicast and broadcast packets. The AXI Ethernet Lite MAC core does not support multicast packets. This field is always provided in the packet data for transmissions and is always retained in the receive packet data.

Source Address

The source address field is 6 bytes in length. This field is always provided in the packet data for transmissions and is always retained in the receive packet data.

Type/Length

The type/length field is 2 bytes in length. When used as a length field, the value in this field represents the number of bytes in the subsequent data field. This value does not include any bytes that might have been inserted in the padding field following the data field. The value of this field determines if it should be interpreted as a length as defined by the IEEE 802.3 standard or a type field as defined by the Ethernet protocol.

The maximum length of a data field is 1,500 bytes. Therefore, a value in this field that exceeds 1,500 (0x05DC) indicates that a frame type rather than a length value is provided in this field. The IEEE 802.3 standard uses the value 1536 (0x0600) or greater to signal a type field. The AXI Ethernet Lite MAC does not perform any processing of the type/length field. This field is transmitted with the least significant bit first but with the high order byte first. This field is always provided in the packet data for transmissions and is always retained in the receive packet data.

Data

The data field can vary from 0 to 1,500 bytes in length. This field is always provided in the packet data for transmissions and is always retained in the receive packet data.

Pad

The pad field can vary from 0 to 46 bytes in length. This field is used to ensure that the frame length is at least 64 bytes in length (the preamble and SFD fields are not considered part of the frame for this calculation) which is required for successful Carrier Sense Multiple Access with Collision Detection (CSMA/CD) operation. The values in this field are used in the frame check sequence calculation but are not included in the length field value if it is used. The length of this field and the data field combined must be at least 46 bytes.

If the data field contains 0 bytes, the pad field is 46 bytes. If the data field is 46 bytes or more, the pad field has 0 bytes. For transmission, this field is inserted automatically by the AXI Ethernet Lite MAC if required to meet the minimum length requirement. If present in the receive packet, this field is always retained in the receive packet data.

FCS

The Frame Check Sequence (FCS) field is 4 bytes in length. The value of the FCS field is calculated over the source address, destination address, length/type, data, and pad fields using a 32-bit CRC defined

$G(x) = x32 + x26 + x23 + x22 + x16 + x12 + x11 + x10 + x8 + x7 + x5 + x4 + x2 + x1 + x0$

The CRC bits are placed in the FCS field with the x31 term in the left most bit of the first byte and the x0 term is the right most bit of the last byte (that is, the bits of the CRC are transmitted in the order x31, x30,..., x1, x0).

The AXI Ethernet Lite MAC implementation of the CRC algorithm calculates the CRC value a nibble at a time to coincide with the data size exchanged with the external PHY interface for each transmit and receive clock period. For transmission, this field is always inserted automatically by the AXI Ethernet Lite MAC core and is always retained in the receive packet data.

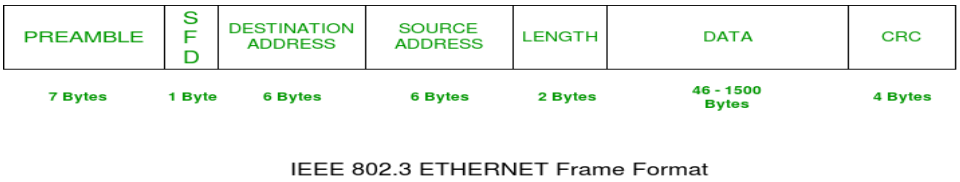
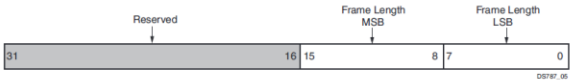


Figure 3. 4 Ethernet Frame Format

Tx Mechanism

- Write Packet in TX buffer
- Write Payload size in TX length register



- Initiate transmission by setting status bit of TX control register

- Poll for status bit for checking completion of TX

Figure 3. 5 Ethernet Transmission sequence

3.2 The comparison test case

3.2.1 Test objective

Comparison of time taken between CDMA and Microblaze based data Transfer.

3.2.2 Test specification

Test Board : KC705

VIVADO 2017.4

SDK 2017.4

Payload size:	1002 Bytes
CDMA Transfer:	BRAM→CDMA→ETHERNET
Microblaze Transfer:	BRAM→MICROBLAZE→ETHERNET

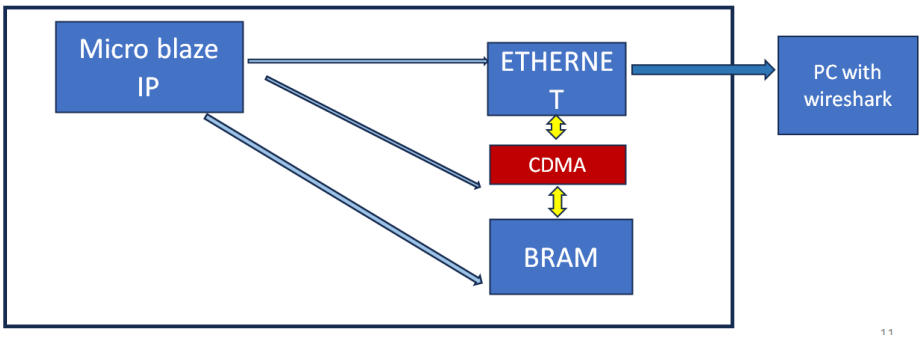


Figure 3. 6 Generic block design for Test case

3.2.3 Test design

In test design 1 ,microblaze was used as data transfer mechanism ,where in test design 2 DMA was used as data transfer mechanism ,though microblaze was also present but for control only.

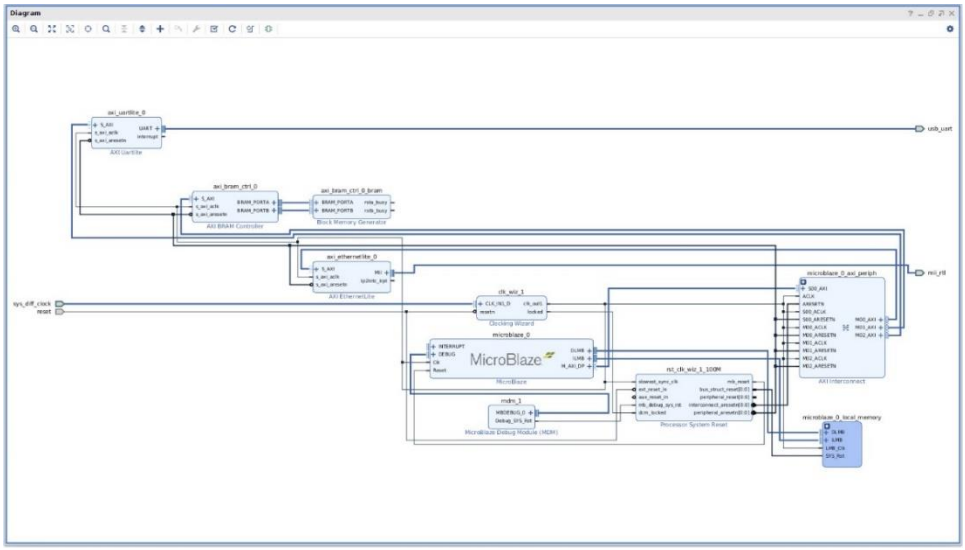


Figure 3. 7 Generic Block design of test case1 (micro blaze data transfer)

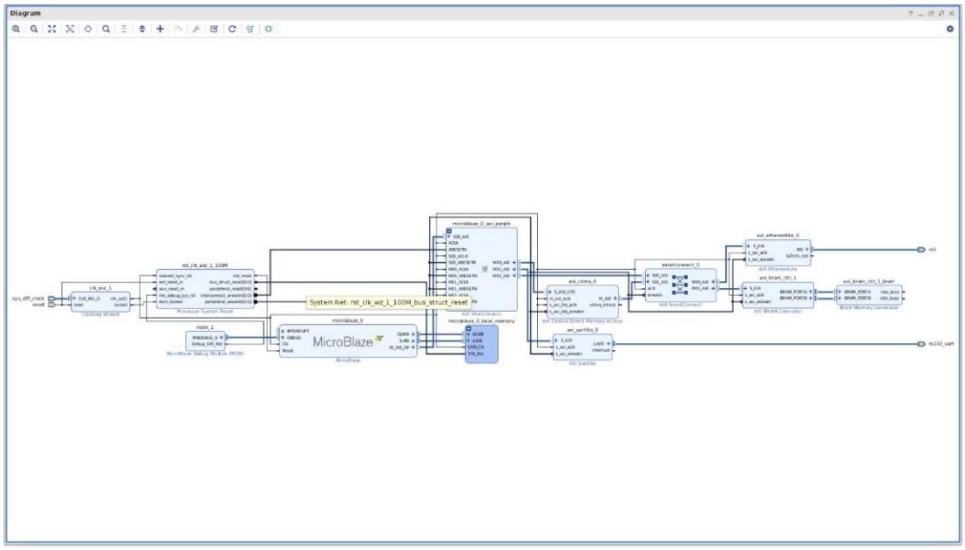


Figure 3. 8 Generic Block design of test case2 (CDMA data transfer)

3.2.4 Challenges

Data Alignment issues: Ethernet has Tx and Rx which is 32 bit 2K aligned memory (expandable upto 4K). This alignment causes issues while storage of 8 bit data in 32bit memory location. figure shows how 4 nos of 8 bit data (data = 0x89h ,0xCBh, 0x45h,0x60h) is expected to be saved in memory and how it is actually being saved to memory.

Table 2: Non aligned and aligned Memory representation

Non aligned memory	Aligned memory
60	89CB4560
45	89CB4560
CB	89CB4560
89	89CB4560

Solution: To overcome this issue, in place of saving actual data in BRAM .the aligned data array was made in microblaze and then it was saved in BRAM.

Failure of DMA simple transfer for complete payload: Once the Aligned data is available simple transfer function was used for sending complete set of data however it was seen that ,for set of 8 location initial 4 location data was correct ,however next 4 locations were filled by 0 .And this happened for each 8 set of location .The problem can be clearly visualized by figure.

Table 3: Memory content after simple DMA Transfer Operation

Expected Ethernet memory	Actual Ethernet memory
89CB4560	0000FC5A
89CB4560	0000FC5A
89CB4560	0000FC5A
89CB4560	0000FC5A
0000FC5A	00000000
0000FC5A	00000000
0000FC5A	00000000
0000FC5A	00000000

Solution : For resolving this issue ,4 set of memory locations were transferred in one simple transfer function inspite of sending complete data ,which transferred the data correctly .

3.2.5 Test results

Table 4: Test Results for Microblaze Vs CDMA Data Transfer Test Case

	Time taken in CDMA transfer (s)	Time taken in microblaze transfer (s)
1 cycle	0.00064335	0.00123197
10 cycle	0.00642884	0.01229624
100 cycle	0.06428354	0.12295754
1000 cycle	0.64283054	1.22957054
2000 cycle	1.28566054	2.45914054

3.2.6 Conclusions

The CDMA based Data transfer method is faster than Microblaze method .However Performance of CDMA is limited by no. of bytes in one operation.

CHAPTER 4

ZynqMPSOC based design

4.1 Literature survey

4.1.1 ZynqMPSOC architecture

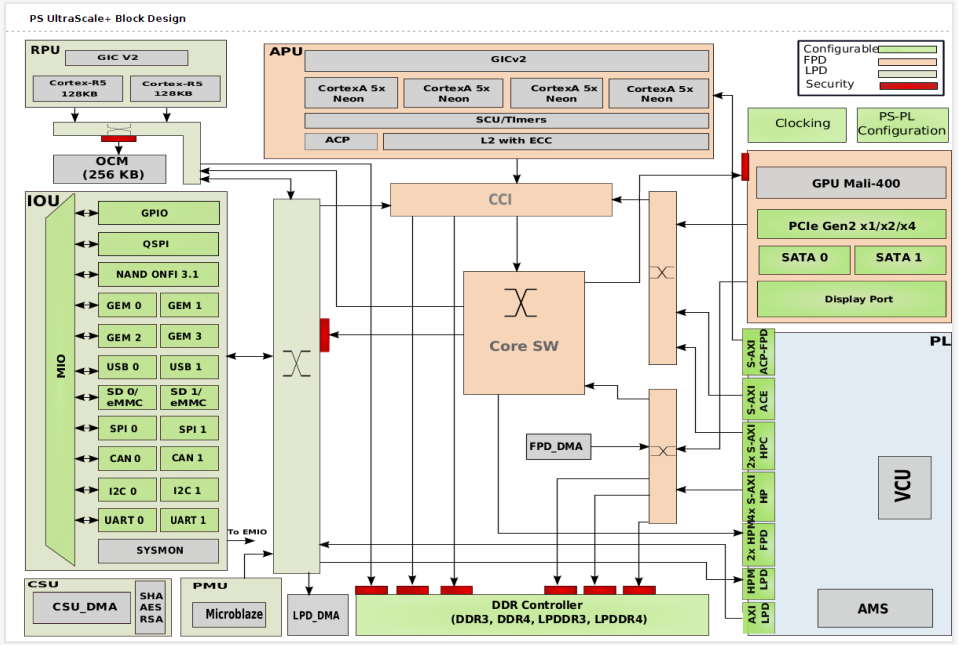


Figure 4. 1 ZynqMPSOC architecture

Processing Units

In the MPSoC, there are two main blocks with different specialized processing units:

Processing System (PS):

APU: Quad or Dual core Cortex-A53 application processing unit. ARM v8 64-bit architecture. It supports:

Asymmetric Multi Processing (AMP): each core running different applications (limited support due to shared HW infrastructure).

Symmetric Multi-Processing (SMP): all of the cores running the same software (e.g. Linux operating system).

RPU: Dual core Cortex-R5 real-time processing unit. ARM v7 32-bit architecture.

Split Mode: each core running different applications as totally independent CPUs.

Lockstep Mode: both cores running the same application for higher security.

PMU: Platform management unit based on triple module redundant Microblaze processor.

CSU: Configuration Security Unit based on triple module redundant Microblaze processor.

GPU: MALI-400 graphic processing unit (available in EG and EV MPSoC families).

Programmable Logic (PL):

VCU: Video control unit with hardware codecs and compression (available in EV MPSoC family).

RF: Radio frequency unit with up to 16 channels RF-ADCs and RF-DACs (available in RFSoC family).

Power Management

The power management in the MPSoC is handled by the Platform Management Unit (PMU).

Power Domains

There are four different power domains in the MPSoC devices:

Low Power Domain (LPD): RPU, PMU, CSU, LPD_DMA, and LPD peripherals

Full Power Domain (FPD): APU, FPD_DMA, and FPD peripherals

PL Power Domain (PLPD): Programmable logic

Battery Power Domain (BPD): Real Time clock and Battery-backed RAM (BBRAM) for secure configuration key.

Each power domain can be individually isolated. The platform management unit (PMU) on the LPD facilitates the isolation of each of the power domains. Additionally, the isolation can be turned on automatically when one of the power supplies of the corresponding power domain is accidentally powered down. Since each power domain can be individually isolated, functional isolation (an important aspect of safety and security applications) is possible. As an application example that we will see later, because the PS and PL resides in two different power domains, the Processing System can be used as a full-featured SoC without powering up the Programmable Logic.

Power Modes

The MPSoC supports three different operational power modes:

Battery Powered Mode: maintain critical information over the time when MPSoC is powered-off.

Low Power Mode: only the devices in the LPD are powered up.

Full Power Mode: all the power domains are activated, including Programmable Logic.

I/O Peripherals

The Zynq UltraScale+ MPSoC features a vast amount of I/O peripherals placed in the different power domains:

Low Power Domain (LPD):

- General Purpose I/O (GPIO)
- Quad SPI Flash Memory (QSPI)
- NAND ONFI 3.1 Controller.
- 4x Gigabit Ethernet MAC
- 2x USB3
- 2x Secure Digital IO (SDIO) for SD / eMMC.
- 2x Serial Peripheral Interface (SPI).
- 2x CAN
- 2x I2C

- 2x UART
- System Monitor

Full Power Domain (FPD):

- PCIe Gen2 x1/x2/x4
- 2x Serial Advanced Technology Attachment (SATA)
- 2x Display Port 1.2 (DP)

Programmable Logic Power Domain (PLPD):

- PCIe Gen3 x16, Gen4 x8.
- 100G Ethernet.
- 150G Interlaken v1.2.
- GTH and GTY Transceivers.

The peripherals' I/O interfaces can be router to the Multiplexed I/O (MIO) and the Extended Multiplexed I/O (EMIO).

- There are up to 78 MIO ports divided in three banks available from the processing system and the MIO itself resides in the Low Power Domain.

- As the number of MIO ports is limited, many of the available peripherals can be routed to the programmable logic through the Extended MIO (EMIO).

AMBA AXI4

The Zynq UltraScale+ MPSoC provides AMBA AXI4 capabilities for high performance data communications.

Interconnect

There are three main AMBA interconnect blocks:

Full Power Domain (FPD):

- AXI Cache-Coherent Interconnect (CCI) Central/Core Switch
- Low-Power Domain (LPD):

LPD Switch

The system provides the following AMBA AXI4 compliant interfaces for PS-PL communications:

Master Interfaces

The PS acts as master and the PL as slave.

- 3x HPM: PS General Purpose Master interfaces (32, 64, and 128 bits width, default 128)

- 2x HPM FPD: From full power domain
- 1x HPM LPD: From low power domain (low latency from peripherals and RPU)

Slave Interfaces

The PS acts as slave and the PL as master-

7x PL General Purpose Master interfaces (32, 64, and 128 bits width, default 128):

- 2x S-AXI HPC FPD: access to full power domain

4x S-AXI HP FPD: access to full power domain and DDR controller

1x AXI LPD: access to low power domain

- 1x S-AXI ACE: PL Master AXI Coherency Extension (ACE) interface for coherent I/O to A53 L1 and L2 cache (128 bits width)
- 1x S-AXI ACP-FPD: PL Master ACP interface for L2 cache allocation from PL masters, limited to 64-byte cache line transfers (128 bits width).

4.1.2 GENESYS -ZU board

The Digilent Genesys ZU is a stand-alone Zynq UltraScale+ MPSoC prototyping and development board. It is an advanced computing platform with powerful multimedia and network connectivity interfaces. The excellent mix of on-board peripherals, upgrade-friendly DDR4, Mini PCIe and microSD slots, multi-camera and high-speed expansion connectors are bound to support a wide number of use-cases. Furthermore, the Genesys ZU is available in two variants with different MPSoC options and additional features for even more flexibility. Differences are highlighted* throughout this document.

The Xilinx Zynq UltraScale+ MPSoC at the heart of the Genesys ZU is a big leap from the Zynq-7000 series. Faster and more processor cores, upgraded memory interface, integrated gigabit transceivers bring support for DDR4, USB Type-C 3.1, PCIe, SATA, DisplayPort, SFP+* and HDMI*. The Genesys ZU is primarily targeted towards Linux-based applications that allows easy access to Wi-Fi, cellular radio (WWAN), SSD, USB SuperSpeed and 4K video. The bundled microSD card includes an out-of-box demo that boots a Linux image built in Petalinux and includes some test scripts for some of the peripherals.

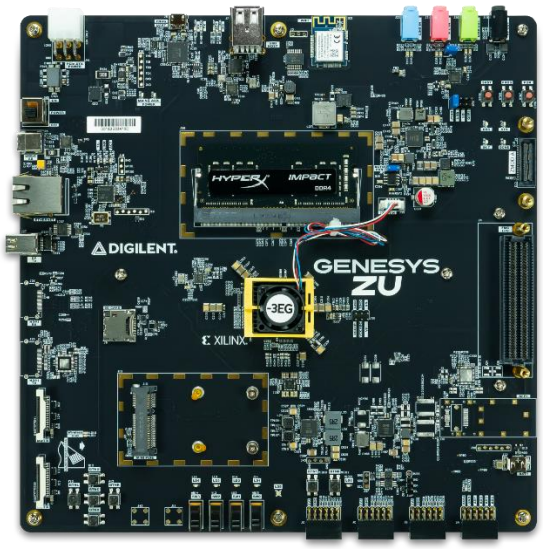


Figure 4. 2 Picture of Genesys ZU-5EV board

Feature group	Sub-feature	Genesys ZU-3EG	Genesys ZU-5EV
Processor	APU	Quad A53	
	RPU	Dual R5	
	Main Memory	DDR4, 4GB, 1866 MT/s, upgradeable	DDR4, 4GB, 2133 MT/s, upgradeable
	GPU	✓	
	Video Codec	✗	✓
Programmable logic	# of logic cells	154K	256K
Peripheral connectivity	USB Type-C 3.1 Gen1 Dual-Role Device	✓	
	MiniPCIe / mSATA dual slot	Half-/Full-size	
	USB 2.0 Host	2 x Type-A	
Network connectivity	On-board Wi-Fi	2.4GHz	
	Ethernet 1G w/ IEEE 1588	✓	
	WLAN / WWAN / LoRa	option - MiniPCIe	
	SFP+ 10G Ethernet	✗	✓
Storage	SD	104 MB/s	
	SSD	option - mSATA	
	Flash	ISSI 256 Mib SNOR	
Multimedia	DisplayPort	1.2a Dual-Lane	
	Pcam	2 x Dual-Lane	
	HDMI Source	✗	✓
	HDMI Sink	✗	✓
	Audio Codec	✓	
Expansion	Low speed - Pmod	4 x	
	Mid-speed - SYZYGY	✓	
	Mid-speed - FMC	✓	
	High-speed - FMC Gigabit	✗	✓
User I/O	LED, Buttons, Switches	✓	

Figure 4. 3 Hardware resource of Genesys ZU-Board

4.1.3 PetaLinux

Introduction

Petalinux, an open-source embedded Linux development solution, has gained significant traction in the realm of embedded systems. Developed by Xilinx, Petalinux offers a comprehensive toolchain tailored specifically for Xilinx's FPGA and SoC platforms. Its primary objective is to simplify the process of creating, customizing, and deploying Linux-based systems on Xilinx devices, thereby accelerating embedded system development.

Comparison with Linux

While Petalinux shares similarities with traditional Linux distributions, it offers several distinct advantages tailored to embedded system development:

Customization and Optimization: Petalinux streamlines the configuration process, allowing developers to customize Linux distributions specifically for their target embedded platforms. This level of customization ensures that the final system is optimized for performance, resource utilization, and power efficiency.

Integration with Xilinx Tools: Petalinux seamlessly integrates with Xilinx's suite of development tools, including Vivado Design Suite and SDK (Software Development Kit). This tight integration simplifies the development workflow, enabling seamless transition from hardware design to embedded software development.

Hardware Abstraction: Petalinux abstracts the underlying hardware complexity, allowing developers to focus on application-level development without worrying about low-level hardware intricacies. This abstraction layer enhances portability and facilitates code reuse across different Xilinx platforms.

Streamlined Development Process: Petalinux provides a unified development environment equipped with essential tools, libraries, and utilities required for embedded system development. This streamlined workflow reduces development time and effort, enabling faster time-to-market for embedded products.

Comprehensive Documentation and Support: Xilinx offers extensive documentation, tutorials, and community forums dedicated to Petalinux development. This wealth of resources empowers developers to troubleshoot issues, explore advanced features, and collaborate with peers within the embedded systems community.

In summary, while Petalinux shares the foundational principles of Linux, its specialized toolchain, integration with Xilinx hardware, and focus on embedded system requirements distinguish it as a preferred choice for FPGA and SoC-based development projects.

Applications

Petalinux Applications in Embedded Systems

Petalinux finds wide-ranging applications across various embedded system domains, including but not limited to:

Industrial Automation: Petalinux is utilized in industrial automation systems for tasks such as process control, monitoring, and data acquisition. Its real-time capabilities, coupled with support for industrial communication protocols, make it well-suited for demanding industrial applications.

Internet of Things (IoT): Petalinux powers IoT devices by providing a flexible and customizable platform for developing edge computing solutions. Its support for wireless connectivity standards, security features, and low-power optimization makes it ideal for IoT device development.

Automotive Electronics: Petalinux is employed in automotive electronics for applications ranging from infotainment systems to advanced driver-assistance systems (ADAS). Its ability to interface with automotive peripherals and sensors, coupled with its reliability and performance, makes it a preferred choice in the automotive industry.

Telecommunications: Petalinux is used in telecommunications equipment for tasks such as network routing, packet processing, and protocol implementation. Its scalability, high-performance networking stack, and support for virtualization enable the development of robust and scalable telecom solutions.

Medical Devices: Petalinux is deployed in medical devices for applications such as patient monitoring, diagnostic imaging, and medical instrument control. Its reliability, real-time capabilities, and compliance with medical standards make it well-suited for mission-critical medical applications.

These examples highlight the versatility and applicability of Petalinux across diverse embedded system domains, showcasing its effectiveness in addressing the unique requirements of each application.

Basic Petalinux Commands

petalinux-create: Command to create a new Petalinux project.

petalinux-create --type project --template <template_name> --name <project_name>

petalinux-config: Command to configure the Petalinux project.

petalinux-config -c <configuration_name>

petalinux-build: Command to build the Petalinux project.

petalinux-build

petalinux-package: Command to package the built images for deployment.

petalinux-package --boot --force --fsbl <fsbl_file> --fpga <fpga_bitstream_file> --u-boot

petalinux-boot: Command to boot the Petalinux image on the target device.

petalinux-boot --jtag --prebuilt 3 --fpga --bitstream <fpga_bitstream_file> --kernel --image <image_file> --dtb <device_tree_file>

Facilities Provided by Petalinux for Embedded System Design

Petalinux facilitates embedded system design through the following features:

1. Hardware Abstraction Layer (HAL): Petalinux abstracts hardware complexity, enabling developers to focus on application development without worrying about low-level hardware details.

2. Customization and Configuration: Petalinux provides tools to customize and configure Linux distributions tailored to specific embedded platforms, ensuring optimized performance and resource utilization.

3. Integration with Xilinx Tools: Petalinux seamlessly integrates with Xilinx's development tools, such as Vivado Design Suite and SDK, streamlining the development workflow from hardware design to software development.

4. Real-time Capabilities: Petalinux offers real-time features and optimizations, making it suitable for applications requiring deterministic response times, such as industrial automation and automotive electronics.

5. Security Enhancements: Petalinux incorporates security features and best practices to safeguard embedded systems against potential threats, ensuring data integrity and system reliability.

6. Community Support and Resources: Xilinx provides extensive documentation, tutorials, and community forums dedicated to Petalinux development, empowering developers with resources to troubleshoot issues and explore advanced features.

7. Power Management: Petalinux incorporates power management features to optimize energy consumption, prolong battery life, and enhance the efficiency of embedded devices.

8. Peripheral Support: Petalinux provides comprehensive support for interfacing with a wide range of peripherals and external devices, enabling seamless integration of hardware components into embedded systems.

9. Cross-Compilation Support: Petalinux supports cross-compilation, allowing developers to build and deploy software for target platforms with different architectures, enhancing portability and flexibility.

10. Debugging and Profiling Tools: Petalinux offers debugging and profiling tools to identify and resolve software issues, optimize performance, and fine-tune embedded applications for optimal efficiency.

11. Update and Maintenance Mechanisms: Petalinux includes mechanisms for software updates and maintenance, ensuring that embedded

systems remain up-to-date with the latest features, patches, and security fixes.

12. Scalability and Flexibility: Petalinux is highly scalable and flexible, accommodating a wide range of embedded system requirements, from resource-constrained IoT devices to high-performance industrial control systems.

4.2 Transforming existing design to Lab-on a chip

4.2.1 Existing design - Distributed architecture

Figure shows existing design of Molecular dynamics cycle. It has two functional module namely MD calculations and Verlet algorithm ,implemented of two different platforms namely FPGA and PC respectively .These two functional module communicate with each other via some communication protocol i.e UART /ethernet .The MD calculation functional module takes x,y,z coordinates from Verlet algorithm via these communication protocol and Calculates the Forces value using Custom IP implemented in FPGA .On receiving forces values the verlet algorithm processes it and generates new set of x,y,z coordinates and this cycle continues .

This distributed design uses communication protocol which takes significant amount of time .

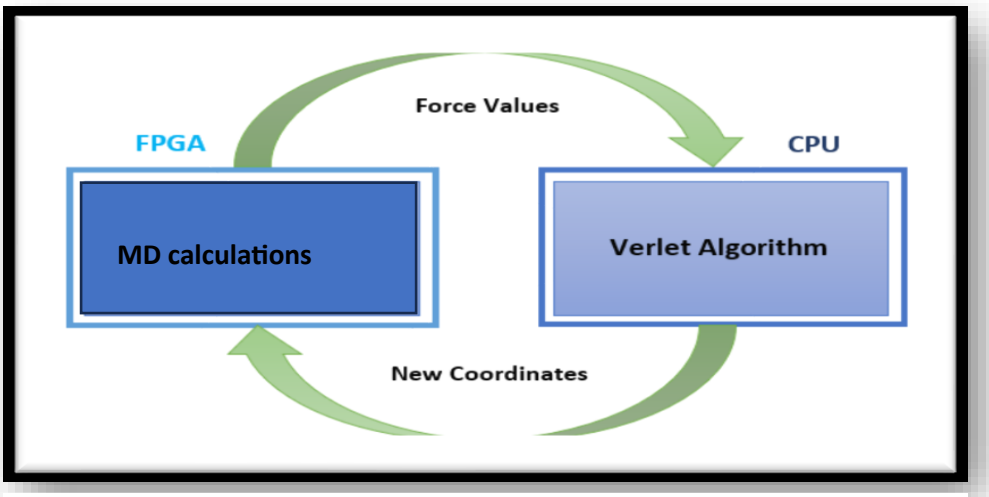


Figure 4. 4 Distributed Design of MD cycle

Figure 4. 5 Distributed Design of MD cycle

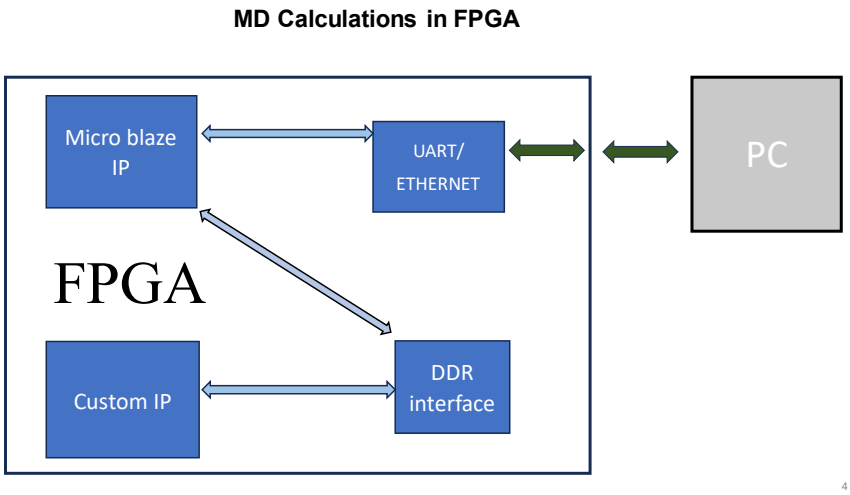


Figure 4. 6 IAP/MD calculations implemented inside FPGA

4.2.2 Proposed Design: Lab-on-a-chip

The Lab -on-a-chip brings the two functional module to single platform hence making the system more compact.This rules out the usage of communication protocol hence saving crucial time incurred by the protocol and hence system becomes faster .

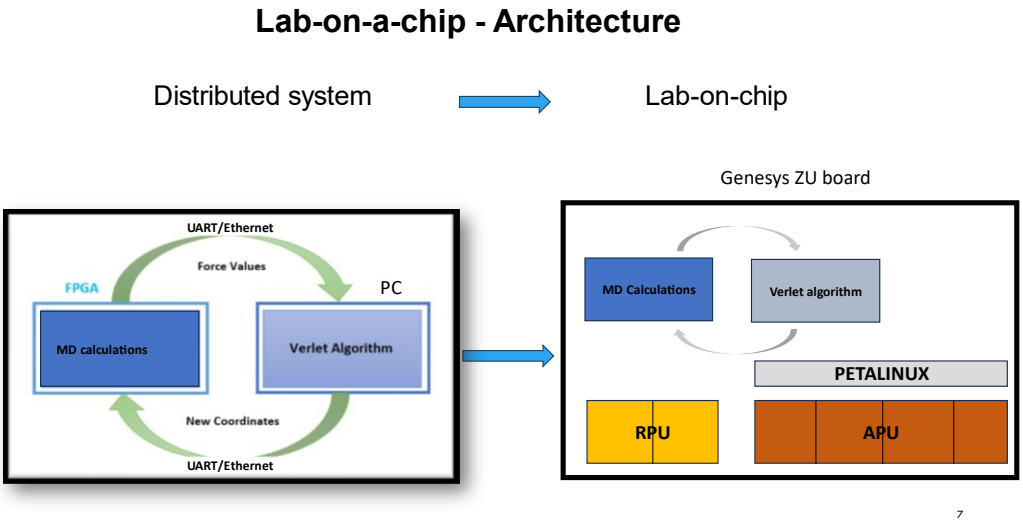


Figure 4. 7 Lab-on-a-chip architecture

4.2.3 The architecture of Lab-on-chip

The ZynqMPSOC facilitates two kind of multiprocessor design namely Symmetric Multiprocessing (SMP) and Asymmetric Multiprocessing (AMP) .The present lab-on-a chip design used AMP which provides facility to implement applications on different processors .

The Verlet algorithm is set of files written in Fortran .It need to be run on OS with appropriate compiler .Hence it is implemented on Petalinux OS running on Application Processing Unit (APU).

MD calculations which is a bare metal application need to be directly implemented over hardware and hence its implemented on Realtime Processing Unit (RPU).The two modules communicate with each other via interprocess-communication mechanism “Shared memory” .

4.2.4 Challenges

The lab-on-a-chip design has been classified into 3 major design challenges

1. Communication between Verlet algorithm and MD application
2. Implementation of Verlet algorithm on PetaLinux
3. Running of MD calculation from PetaLinux

The challenges and solutions to overcome the challenges has been discussed below.

1. Communication between Verlet algorithm and MD application

Introduction: In earlier design, The communication was done by inter device communication protocols like UART or ethernet .No since both the functional modules are implemented on same platform,this protocols are no more required and it will be replaced by simpler inter-process protocol like shared memory .

Shared memory protocols may be used when two process are running on same platforms .In the present design ,it will be used as communication medium for communication between two application

running on different cores of same hardware platform .The following test case demonstrates that how the shared memory can be viable option for communication between two application running on two different cores of same processor.

Test Case: Shared Memory as Inter process communication protocol

Test Objective: To demonstrate the functionality of 2 application running on two different cores of same platform and communicating via shared memory.

Test specification:

Test board : Zybo (zynq-7000 series)

Software tools : Vitis 2023.2

Vivado 2023.2

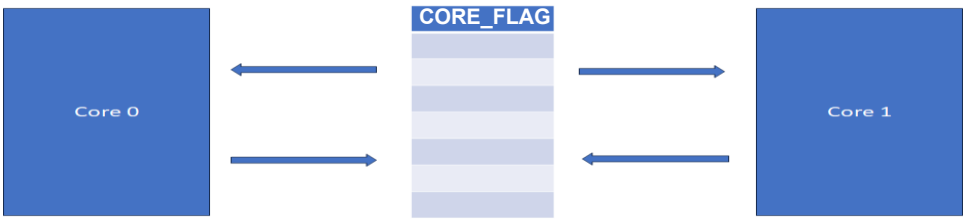


Figure 4. 8 Shared memory

Test Case description:

Zybo board : The ZYBO (ZYnq BOard) is a feature-rich, ready-to-use, entry-level embedded software and digital circuit development platform built around the smallest member of the Xilinx Zynq-7000 family, the Z-7010. The Z-7010 is based on the Xilinx All Programmable System-on-Chip (AP SoC) architecture, which tightly integrates a dual-core ARM Cortex-A9 processor with Xilinx 7-series Field Programmable Gate Array (FPGA) logic. When coupled with the rich set of multimedia and connectivity peripherals available on the ZYBO, the Zynq Z-7010 can host a whole system design. The on-board memories, video and audio I/O, dual-

role USB, Ethernet, and SD slot will have your design up-and-ready with no additional hardware needed. Additionally, six Pmod ports are available to put any design on an easy growth path.

Source code design: A particular region of memory is designated as shared region which can be parallelly accessed by application running on both the cores. This memory region can be designated by editing in Linkerscript file in vitis .One of the memory location is designated as CORE_FLAG. Core 0 initializes this flag as 0 ,both application starts polling for this flag .If this CORE_FLAG=0 ,CORE0 application will perform some action and CORE1 will wait .After performing action ,CORE0 will update the flag to CORE_FLAG=1 .CORE 1 will start performing action and CORE0 will wait and this cycle will go on . One particular memory location (Other than CORE_FLAG) is chosen and initialized by 0 by CORE0 . Content of this memory location is updated one by one by each core .

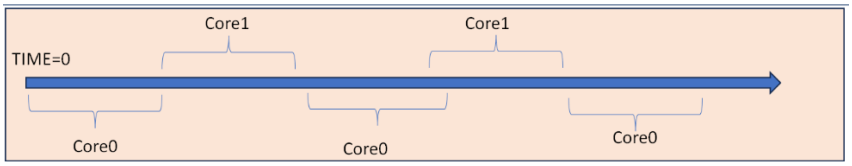


Figure 4. 9 Core sequencing

CORE0 application :

Initialize CORE_FLAG=0

while(1)

{ while(CORE_FLAG==0)

{

- Read the content
- Print it
- Add 4 to it
- Write to same location
- CORE_FLAG==1

}

}

CORE1 application :

```
while(1)
{  while(CORE_FLAG==1)
    {
```

- Read the content
- Print it
- Add 1 to it
- Write to same location
- CORE_FLAG==0

```
    }
}
```

Test Result

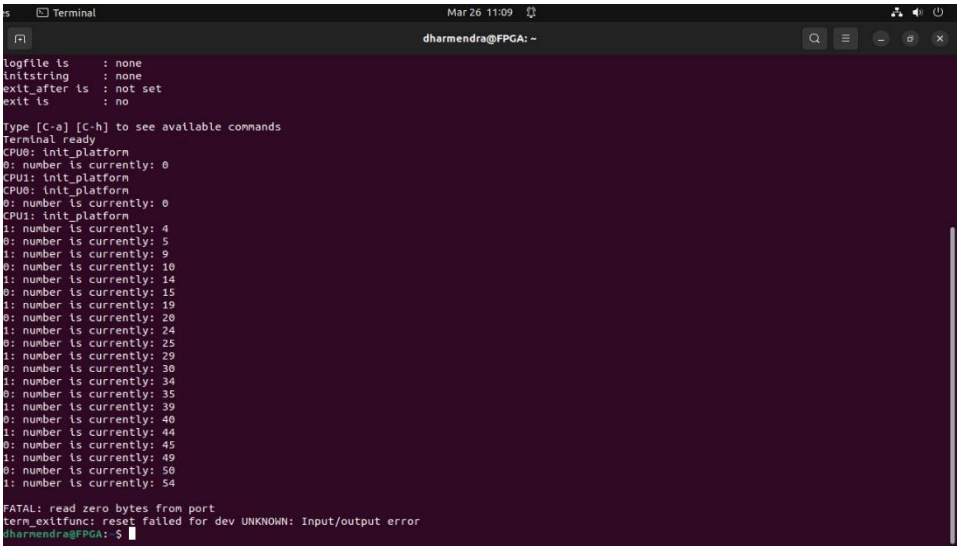


Figure 4. 10 Test result for shared memory test case

Test Result Analysis

The CORE0 initializes the memory content by writing 0 and CORE_FLAG=0. The both core starts polling for CORE_FLAG using while loop. As CORE_FLAG=0 ,CORE0 gets chance to perform action .It reads the memory content and print it then it add 4 to it and write back to same location and update CORE_FLAG=1.Now CORE1 gets the chance to perform action .It reads the memory location and prints it which is correctly printed as 4 .CORE1 adds 1 to it and update CORE_FLAG=0 .Now CORE0 turn comes and it print updated value 5 .This cycle continues .The Result are as expected.

2 Implementation of Verlet algorithm on PetaLinux

Introduction: Verlet algorithm is an algorithm which is responsible for generation of x,y,z coordinates from Force values .This is a set of files .The algorithm is mainly written in Fortran language which need Fortran compiler to compile .The compiler for fortran is “gfortran” .In Distributed architecture of MD cycle ,the verlet algorithm runs on Linux based PC.

The Petalinux is small scale linux which does not have “gfortran” compiler.It is also not possible to install it via package manager .as package manager itself is not available .So the option for having “gfortran” is rules out due to limited resource available in petalinux .

There comes the role of cross compiler . Cross-compilation is the process of compiling software on one platform to run on another platform. Cross-compiling can save time and resources by allowing you to compile code on a more powerful machine, which can be faster than compiling on the host device itself. It also allows one to compile code for platforms that may not have the necessary tools or resources to compile the code themselves. To perform cross-compilation a cross-compiler is needed,

which will generate executable code for the platform other than that in which the compiler is currently running.

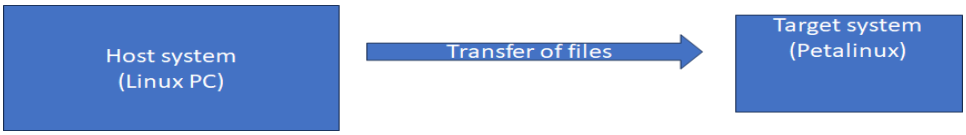


Figure 4. 11 Cross-compilation basics

The cross compiler used here is "aarch64-linux-gnu-gfortran" which generate executable which is compatible with ARM aarch64 series of devices .Following test case is devised to demonstrate that verlet algorithm can be implemented on Petalinux with the help of cross compiler "aarch64-linux-gnu-gfortran".It also covers the time performance comparison between verlet implemented on PC and petalinux .

Test case : Implementation of Verlet algorithm in Petalinux.

Test objective : To implement verlet algorithm in petalinux and comparison of time performance between Petalinux and PC.

Test specification: The fortran code which implement verlet algorithm internally calls C based code which communicates with FPGA for data communication in present design .To eliminate FPGA from loop .This C based code is replaced with another C based loopback code which simply loops backthe data .The objective of this is to take out FPGA board from loop.The objective is not to functionally replace the FPGA board but to measure the time performance of the loop without dwelling much in Accuracy of the data .

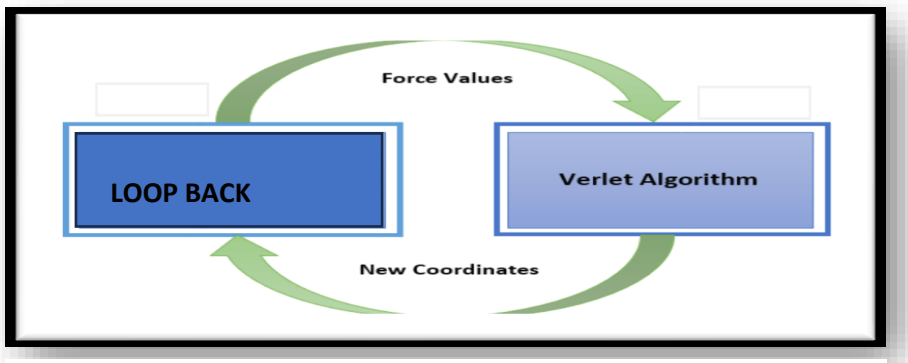


Figure 4. 12 Verlet algorithm with loopback program

Figure 4. 13 Verlet algorithm with loopback program

Test Procedure

1. Updation of Make file : The set of Verlet files include makefile which need to be appropriately updated to include the details of cross compiler .The existing compiler details need to be commented and following cross compiler need to be added in the Makefile.

F77 =/usr/bin/aarch64-linux-gnu-gfortran

LIBS=

F77FLAGS = -C

OPTFLAGS =-O

LIBFLAGS = -crusv

LINKFLAGS = -static-libgfortran

2. Cross compiling loopback code : The C based loopback code also need to be cross compiled using “aarch64-linux-gnu-gcc” to generate its executable which is compatible with ARM processor .

3. Run Make command .This generates executable ./dynamic.x

4. Transfer of complete files to petalinux using SSH: The complete set of files to be transferred to one working directory of petalinux using SSH

5. Running executable dynamic.x in petalinux will start verlet algorithm.It is run here for 500 cycles.

Test Results

Table 5:Test Results for Verlet algorithm implementation in PC and Petalinux

500cycles	Verlet
PC	5.32s
Petalinux	7.05s

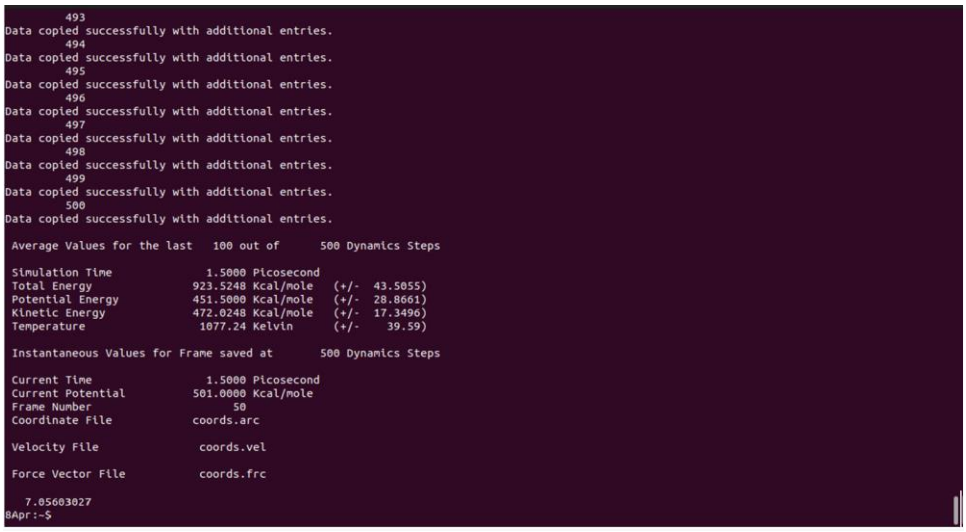


Figure 4. 14 Test Results -Verlet in petalinux

Resul Analysis : (All time measurements are in seconds)

Table 6:Test Result analysis data

500cycles	Verlet	Communication (UART 115200bps)	Custom IP(FPGA)	Total
PC	5.32	153.29	0.8x500=400	558.61
Petalinux	7.05	~0	0.8x500=400	407.05

The total time in a complete MD cycle can be devided into 3 major divisions time taken by verlet ,time taken by Communication and time taken by CUSTOM IP running in FPGA .The total time is summation of all these 3 time. This Verlet algorithm running in petalinux takes 1.73 second more time than PC. However in complete cycle implementation, It can be seen that verlet contributes very less in total time duration. The Total time in petalinux implementation will be lesser than PC due to absence of inter device communication protocol which will save around 153.29s.

3 Running MD calculation from PetaLinux

Introduction : MD calculation is Bare metal application .This is directly implemented on hardware without interface of any OS .Generally Petalinux cannot not implement bare metal application .Hence it was decided to implement it on Real time processing unit (RPU).Once it is implemented on RPU .it was major challenge to run it from Petalinux .This solution was provide by OpenAMP .

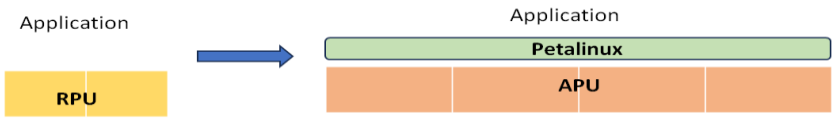


Figure 4. 15 Running RPU application in APU

OpenAMP : OpenAMP (Open Asymmetric Multi-Processing) on Zynq UltraScale+ MPSoC is a framework that facilitates the development of software applications across heterogeneous processing environments. The Zynq MPSoC integrates ARM Cortex-A53 processors with ARM Cortex-R5 real-time processors and FPGA fabric, making it ideal for applications requiring high performance and real-time capabilities. OpenAMP enables

seamless communication and resource sharing between these heterogeneous cores, enhancing parallel processing and improving overall system efficiency. By leveraging OpenAMP, developers can optimize workload distribution, reduce development complexity, and improve scalability

OpenAMP on Zynq UltraScale+ MPSoC consists of several key components that work together to enable efficient asymmetric multiprocessing. These components include the RemoteProc framework, which manages the lifecycle of remote processors, including booting, loading firmware, and handling crashes. The RPMsg (Remote Processor Messaging) protocol facilitates inter-processor communication, providing a standardized method for message passing between the heterogeneous cores. Additionally, the VirtIO framework allows for virtualization of I/O devices, enabling shared access to peripherals and resources. Together, these components enable developers to efficiently utilize the diverse processing capabilities of the Zynq MPSoC, ensuring robust, scalable, and high-performance embedded applications.

OpenAMP on Zynq UltraScale+ MPSoC functions by enabling seamless communication and resource management between its heterogeneous processors. The RemoteProc framework initializes and manages the ARM Cortex-R5 and FPGA-based soft cores, while RPMsg provides a standardized messaging protocol for inter-processor communication. VirtIO facilitates the sharing of I/O devices and resources across these cores. This coordination allows tasks to be distributed efficiently, leveraging the real-time capabilities of the Cortex-R5 processors alongside the high-performance Cortex-A53 processors, resulting in optimized system performance and resource utilization in embedded applications

Test Case & objective : To implement a test application in RPU and run it from Petalinux.

Test specification :

software tools : Vitis 2023.2

Vivado 2023.2

Petalinux 2023.2

Test plaform: Genesys ZU -5EV (ZynMPSOC ultrascale)

Test application code :

```
int main()
{
    init_platform();

    xil_printf("Hello from RPU0 \n\r");

    cleanup_platform();
    return 0;
}
```

Figure 4. 16 Test application for RPU

Test Procedure: Figure shows the complete flow and process of the design.

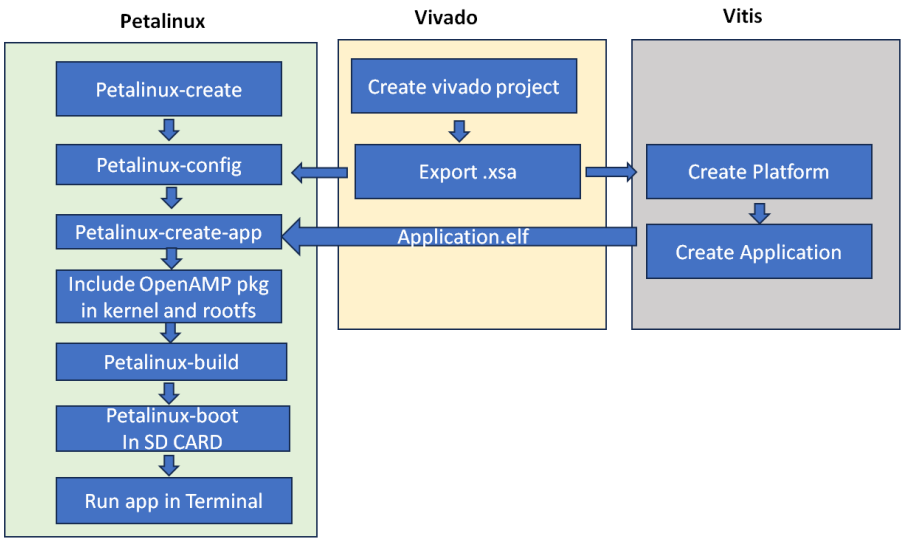


Figure 4. 17 workflow for running RPU application in APU

1 Vivado block design is created. bitstream is generated and exported

2. Vitis platform is created using the exported .xsa file .a new domain of Cortex r5_0 is added in the project .The platform is bult.

3. An application with test code is created .Linker script is updated accordingly .The application is build .This generates executable .elf file .This file wiil be taken into petalinux.

4. A petalinux project is created with ZynMPSOC template .The .xsa file is imported to petalinux via petalinux-config command.

5. Petalinux application is created using .elf file from vitis

6. OpenAMP package is selected in to kernel and rootfs.

7. Petalinux build and SD ard boot is initiated

8. Application is run in Petalinux Terminal.

Test Result

Figure 4. 18 Test Result for running RPU application in APU

Result Analysis

“Hello from RPU0 “ is printed but it is intermixed with PID number .which is being debugged.

CHAPTER 5

Conclusion and Future work

5.1 Conclusion

The thesis work is primarily exploratory in nature which investigates some methods like interrupt-based transmission and DMA based data transfer method to expedite the Molecular Dynamics Calculations and hence making all over Molecular Dynamics Cycle faster. The methods have been validated with test cases and results.

The objective of the Thesis revolves around transforming the Distributed architecture of implementation of molecule dynamics cycle to Lab-On-chip architecture which focusses on implementing two functional modules namely MD calculations and Verlet algorithm which were earlier implemented on two different platforms, to single platform. The Thesis has conclusively proposed an architecture for Lab-On-chip for implementation the same design. The architecture will be basis for implementing this distributed architecture into ZYNQMPSOC based FPGA.

5.2 Future works

5.2.1 Implementation of DMA based Data transfer in Molecular Dynamics Cycle which bypasses Microblaze hence improving efficiency and time performance.

5.2.2 Implementation of Molecular Dynamics Cycle in Lab-on-a-chip architecture using Asymmetric -Multi Processing (AMP) (Using RPU and APU both)

5.2.3 Implementation of Molecular Dynamics in Lab-on-a-chip architecture using Symmetric Multi-Processing (SMP). (Using APU only.)

BIBLIOGRAPHY

1. Satya S Bulusu, Srivathsan Vasudevan, “FPGA Accelerator for Machine Learning Interatomic Potential-Based Molecular Dynamics of Gold Nanoparticles”, in IEEE Access, Volume 10, pp 40338-40347, April 2022.
2. AXI UART Lite v2.0 Product Guide (PG142)
3. Basys-3 Reference Manual https://digilent.com/reference/programmable-logic/basys-3/reference-manual
4. https://github.com/Xilinx/embeddedsw/tree/master/XilinxProcessorIPLib/drivers/uartlite
5. AXI Ethernet Lite MAC v3.0 Product Guide (PG135)
6. AXI Central Direct Memory Access v4.1 Product Guide (PG034)
7. Microblaze Processor Reference Guide UG984
8. PetaLinuxTools Documentation: Reference Guide (UG1144)
9. Xilinx Zynq UltraScale+ MPSoC Technical Reference Manual
10. Zynq UltraScale+ MPSoC Data Sheet: Overview (DS891)
11. Zynq UltraScale+ MPSoC Data Sheet: DC and AC Switching Characteristics (DS925)
12. Zynq UltraScale+ MPSoC: Embedded Design Tutorial (UG1209)
13. Performance Tuning and Optimization on Zynq UltraScale+ MPSoC (XAPP1265)
14. Zynq UltraScale+ MPSoC Software Developer Guide (UG1137)
15. Libmetal and OpenAMP User Guide UG1186 (v2023.2) November 7, 2023