

**DATA TRANSFER PERFORMANCE ANALYSIS FROM
PROGRAMMABLE LOGIC TO PROCESSING SYSTEM OF ZYNQ 7000**

by

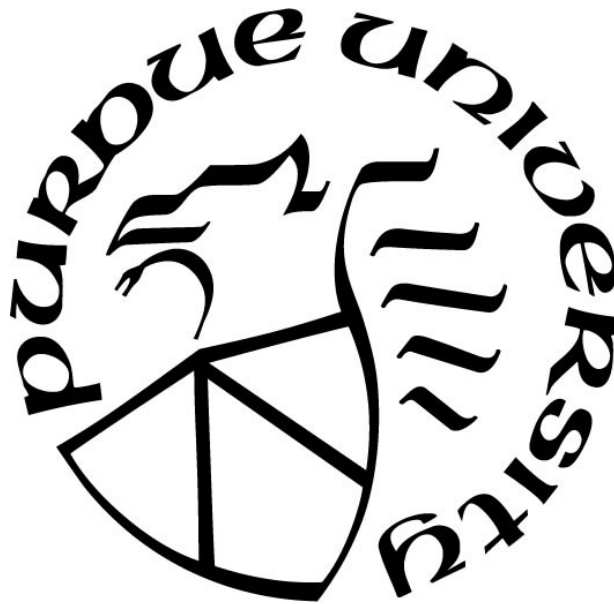
Tilottoma Barua

A Thesis

Submitted to the Faculty of Purdue University

In Partial Fulfillment of the Requirements for the degree of

Masters of Science in Engineering



Department of Electrical and Computer Engineering

Fort Wayne, Indiana

August 2020

THE PURDUE UNIVERSITY GRADUATE SCHOOL
STATEMENT OF COMMITTEE APPROVAL

Dr. Todor Cooklev, Chair

Harris Professor of Wireless Communications and
Applied Research Director of the Center for Wireless Technology

Dr. Chao Chen

Director of Engineering Graduate Program in ECE Department
Associate Professor of Computer Engineering

Dr. Yanfei Liu

Associate Professor of Electrical and Computer Engineering

Approved by:

Dr. Chao Chen

Head of the Graduate Program

ACKNOWLEDGMENTS

At first, I would like to thank Dr. Todor Cooklev for allowing me to proceed with my MSc. thesis under his supervision. I found him supportive during the whole journey while working with him. I am grateful to him for being my supervisor because when I work under him I always felt motivated and happy. My special gratitude goes to the members of my MSc committee, Dr. Chao Chen and Dr. Yanfei Liu for accepting their role. They always helped me by reading my dissertation and providing useful feedback. I would like to thank the Electrical and Computer Engineering Department of Purdue Fort Wayne for giving me this wonderful opportunity to complete the thesis, and for providing the necessary support in the school. Finally, I would like to take the opportunity to thank my parents and my family for their tremendous support during my school time. I would like to thank my husband Anomadarshi Barua, for his continuous support and encouragement throughout my graduate studies at Purdue University Fort Wayne.

TABLE OF CONTENTS

LIST OF TABLES.....	6
LIST OF FIGURES	7
LIST OF ABBREVIATION	9
ABSTRACT.....	10
1. INTRODUCTION	11
1.1 Motivation.....	12
1.2 Why Performance Analysis is Needed - Data Movement Challenges:	13
1.3 Application Areas of ZYNQ-7000-“Data Flow”:.....	14
1.3.1 Video Processing of Drones	14
1.3.2 LiDAR in Autonomous Vehicle	15
1.4 Thesis Outline	16
2. BACKGROUND AND RELATED WORK	18
2.1 Background:	18
2.1.1 Hardware-Software co-design	18
2.2 Related Works with Hardware/Software Co-Design:	19
2.3 Related Works with ZYNQ 7000:	21
2.4 Some More Work On A Different Platform	22
2.4.1 HW/SW Co-design for Exascale System	22
2.4.2 Coarse-Grained Reconfigurable Accelerators (CGRA Architecture)	23
2.4.3 Different Embedded System Platform for Co-Design Applications	23
2.4.4 FPGA and other SoC Platforms.....	24
3. HARDWARE-SOFTWARE PLATFORM AND METHODOLOGY	26
3.1 ZYNQ-7000 Architecture	26
3.1.1 Overview.....	26
3.1.2 ZYNQ-7000 Features	28
3.1.3 ZYNQ-7000 Communication Interfaces	29
3.1.4 Processing System and Programmable Logic Interfaces.....	34
3.1.5 Theoretical Data Transfer Throughput of PL-PS Interfaces.....	35
3.2 Software Platforms.....	36

3.2.1	Vivado Design Suite	36
3.2.2	Software Development Kit (SDK).....	37
3.3	Methodology	38
4.	DESIGN AND IMPLEMENTATION	39
4.1	Hardware-Software Design Co-simulation process.....	39
4.2	Hardware Design	41
4.2.1	Designing “Sample Data Generator” using RTL Flow:	41
4.2.2	AXI4-Stream interface signals	43
4.2.3	Sample Data Generator- Testbench Logic Simulation	44
4.2.4	Hardware Design with VIVADO Design Suite.....	46
4.3	Software Design.....	51
5.	EXPERIMENTAL RESULTS AND PERFORMANCE ANALYSIS	54
5.1	Experimental Setup.....	54
5.1.1	Power Measurement in Zedboard.....	55
5.2	Performance Analysis For Different Data Length.....	56
5.3	Energy Consumption	58
5.4	Performance Analysis For Different Burst Size and PL Clock Frequency	59
5.4.1	Analysis	64
6.	CONCLUSION AND FUTURE WORK	65
6.1	Conclusion	65
6.2	Future Work	66
	APPENDIX A.....	67
	APPENDIX B.....	70
	REFERENCES	72

LIST OF TABLES

Table 3.1: AXI4 Feature Availability and IP Replacement.....	33
Table 3.2: Theoretical Data Transfer Throughput of PL-PS Interfaces.....	36
Table 4.1: ZYNQ Memory Map	51
Table 5.1: Data Transfer Performance Analysis for GP,1-HP and ACP ports	57
Table 5.2: Power Consumption during data transfer	58
Table 5.3: Energy Consumption	59
Table 5.4: Performance Analysis of High-performance port(HP0) For Different Burst Size and PL Clock Frequency	61
Table 5.5: Performance Analysis of Accelerator Coherency Port (ACP) For Different Burst Size and PL Clock Frequency.....	62
Table 5.6: Performance Analysis of General Purpose Port (S_GP0) For Different Burst Size and PL Clock Frequency	63

LIST OF FIGURES

Figure 1-1: A simplified model of Zynq.....	12
Figure 1-2: LiDAR in Autonomous Vehicle[33].....	15
Figure 1-3: A flow graph of thesis outline.....	17
Figure 2-1: FPGA and CPU Integration[11].....	21
Figure 2-2: A timeline of reconfigurable computing system evolution[31]	25
Figure 3-1: Zynq Block Diagram[18]	26
Figure 3-2: Zedboard	27
Figure 3-3: Zynq-7000 Architecture[18]	28
Figure 3-4: System-Level Address map[17].....	29
Figure 3-5: Read and Write Channels[17].....	30
Figure 3-6: Two-way VALID/READY Handshake[17].....	31
Figure 3-7: AXI4-Stream Transfer with Unidirectional Channel[17]	32
Figure 3-8: Zynq Processing System Architecture[18]	34
Figure 3-9: Hardware-software design tools.....	38
Figure 4-1: Hw/Sw Co-design Flow	40
Figure 4-2: The sample data generator AXI stream module.....	42
Figure 4-3: Clock cycle and counter circuit for sample data generator.....	42
Figure 4-4: Data Generation Circuit	43
Figure 4-5: AXI4 stream signals.....	44
Figure 4-6: The sample data generator test bench simulation showing the logic simulation wave diagram	45
Figure 4-7: Basic Hardware Block Diagram	46
Figure 4-8: VIVADO hardware design with high-performance interfacing port and AXI DMA	49
Figure 4-9: ZYNQ memory mapping for the custom design interfaces (HP, ACP, and GP ports) in VIVADO.....	50
Figure 4-10 : Flow chart for the data transfer from PL to DDR memory.....	53
Figure 5-1: Experimental Set up and the Tera Term Serial COM port	55
Figure 5-2: Current sensor shunt resistor on Zedboard	56

Figure 5-3: Performance Analysis of three AXI ports in transferring data from PL to PS	57
Figure 5-4: Tera Term COM console	60
Figure 5-5: Graphical analysis of HP0 interface port For Different Burst Size and PL Clock Frequency.....	61
Figure 5-6: Graphical analysis of ACP interface port For Different Burst Size and PL Clock Frequency.....	62
Figure 5-7: Graphical analysis of S_GP0 interface port For Different Burst Size and PL Clock Frequency.....	63
Figure 6-1: Proposed workflow of video processing.....	66

LIST OF ABBREVIATION

ACP	Accelerator Coherency Port
ARM	Advanced RISC Machines
ASIC	Application Specific Integrated Circuit
AXI	Advanced eXtensible Interface
CPU	Central Processing Unit
DDR3	Double Data Rate Synchronous Dynamic RandomAccess Memory
DMA	Direct Memory Access
DRAM	Dynamic RandomAccess Memory
FPGA	Field Programmable Gate Array
HLS	High Level Synthesis
HP	High Performance
IP	Intellectual Property
LiDAR	Light Detection And Ranging
SoC	System On Chip
GP	General Purpose
SDK	Software Development Kit
PL	Programmable Logic
PS	Processing System
LUT	Look Up Table
OCM	On-Chip Memory

ABSTRACT

Field-Programmable Gate Arrays (FPGAs) were invented in the 1980s. Since then the use of FPGAs in many fields has been growing rapidly. Due to the inherent reconfigurability and relatively low development cost FPGA technology has become one of the important components in data processing and communication systems.

The recent development of computing technology affects not only the software but also requires integrating and utilizing a custom logic design on a dedicated hardware platform.

In this context, this research work analyses and compares on-chip interfaces for hardware/software communications in the Zynq-7000 all programmable SoC-based platform. Several experiments were carried out to evaluate the performance of data communication between the processing system and the programmable logic through general-purpose (GP), high-performance ports (HP), and accelerator coherency port (ACP); the experiments were conducted for bare-metal standalone applications. The results identified the most effective interfaces for transferring data from the PL to PS and store the data to DRAM memory. The Xilinx Software Development Kit (SDK) and Vivado Design Suite together provide hardware/software development platform to evaluate their performance.

One conclusion of this work is that the selection of suitable ports depends on application requirements. For low-bandwidth applications the GP port is appropriate. For high-speed applications, the High Performance (HP) port and Accelerator Coherence Port (ACP) are suitable and work better. The results of this thesis are useful in high-performance embedded systems design.

1. INTRODUCTION

FPGAs continue to have a remarkable impact on computing technology and various fields in engineering. The inherent reconfigurability of FPGAs and comparatively cheap development cost has made it an important tool for research. Nowadays the availability of field-configurable micro-chips that combine multi-core processors and reconfigurable logic incorporated with several DSP slices and block memories has made this path smoother. Such integration leads to the ease of communication system between two different types of processors allowing the systems-on-chip to reduce the complexity of computation and communication. The Zynq-7000 all programmable system-on-chip (APSoC) device from Xilinx integrates a dual-core processing unit running on different software and programmable logic which can be customized to develop different hardware-accelerated systems using different logic combinations and computations, and which further allows interfaces enabling interactions and data exchange between the dual-core processing system using software and hardware components. These cost-effective devices permit complete solutions for embedded systems to be integrated on a chip. To achieve the best and optimal performance of this type of platform, the fastest and best communication channel is required between two different processors. The interfaces between the Processing system (PS) and Programmable Logic (PL) are supported by different intellectual property (IP) cores and the AXI interface system bus. The combination of architectural and technological advances has enabled Zynq-7000 to open a new era in the development of highly optimized computational systems with enormous variations of practical applications. These applications not only include high-performance computing but also data, signal, and image processing with embedded systems. On this contrary, the potential methods need to be studied to decide which method is better for data communications. This type of comparison study has been shown in this thesis with different test cases for Zynq-7000 APSoC. The design flow includes the development of hardware-accelerated tools in the PL supported by available Xilinx IP cores and customized design of IP cores with available features in VIVADO and software development tools (SDK) in the PS for the applications on the bare metal operating system. Bare metal application can be defined as the operating system, which is operational directly on the hardware with no underlying specification. Bare metal application typically has a limited bootloader to start the processor, time, and memory spaces and switch into the main program.

1.1 Motivation

A system-on-chip (SoC) like the ZYNQ -7000 series consists of all the necessary parts and peripherals (e.g. processing units (PS & PL), peripheral interfaces (UART, JTAG, USB, Ethernet, etc.), memory systems, clocking circuit units, Interrupt systems and input/output). Xilinx Zynq-7000 family is the first All Programmable System-On-Chip - APSoC system which is a combination of the dual-core ARM Cortex™ MPCore™-based processing system (PS) and programmable logic (PL) on the same microchip [1].

The processors were used to connect with a Field Programmable Gate Array (FPGA) via remote communication method which made communication between the Programmable Logic (PL) and Processing System (PS) more complicated before the development of the ZYNQ-7000. The Zynq all programmable (SoC) is the latest generation of Xilinx's all-programmable System-on-Chip (SoC) families which combines a dual-core ARM Cortex-A9 with programmable logic (FPGA). To establish communication between the different components, specifically between PS & PL, the Zynq architecture is based on the Advanced extensible Interface (AXI) standard, which provides high bandwidth and low latency connections [1].

The PL(FPGA) part of Zynq is ideal for implementing high-speed logic, arithmetic, and data flow subsystems. On the other hand, the dual-core PS (Processing System) supports software routines and operating systems which ensures the overall functionality of the designed system. The PS-PL interface consists of all the signals available for the designer to integrate the PL-based functions and the PS [2]. A typical simplified model of Zynq is shown in Figure 1-1.

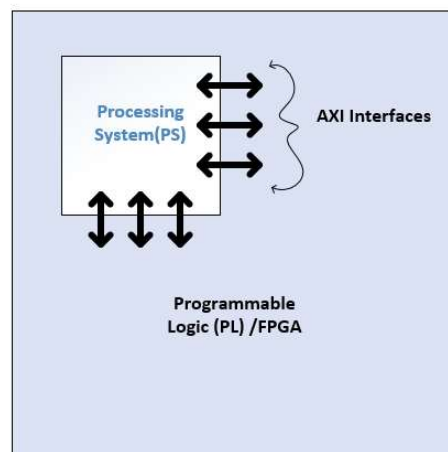


Figure 1-1: A simplified model of Zynq

1.2 Why Performance Analysis is Needed - Data Movement Challenges:

Microelectronics has recently been exponentially progressed with the rapid growth of field-programmable gate array technology (FPGA). This growth has contributed to a range of highly promising applications. Also, adding more transistors on the FPGA chip allows the vast amounts of parallel calculations. Secondly, recent developments in high-speed transceivers permit the transfer of data into and out of FPGAs in high bandwidths. Further, FPGA technology supports computerized algorithms, such as those with LiDAR(Light Detection and Ranging) systems or middleware communication systems that can be used with several FPGA components to accelerate hardware efficiently.

Using a standard interface like AXI, data movement in a System-on-a-Chip (SoC) like Zynq can be on-chip from one usable block to another or off-chip. However, Advanced eXtensible Interface (AXI) protocol, has been adopted by Xilinx as the IP cores. An AXI interface can be defined as an end-to-end linkage between master and slave clients inside the system for transmitting data, addresses, and handshaking signals. Nonetheless, data transfer between normal or memory interfaces are normally performed as on-chip communication. The big issue for a hardware engineer is to determine the appropriate data transfer method for a memory subsystem. Although small data transfer on the chip can be performed using the software instructions, it can be done more effectively with special data transfer tools when large data transfers are made.

The communication overheads have to be evaluated to assess the potential performance increase that can be accomplished with hardware accelerators. Data can be exchanged across a wide range of ports and the right approach for faster data transmission is preferred. The amount of data to be transmitted, the communication bus configuration, and the sharing of the memory of the ARM cache are a key factor in assessing the analysis. Multi-processor approaches to improve system performance are pursued by engineers. Moreover, the processors also need to compute several tasks in parallel. Hence, without an effective mechanism for communication, the purpose or the idea of the development of the multiple-processor system will be degraded.

The right bus protocol and data widths are required for each peripheral on the SoC to optimize the use of on-chip bus structures, that reduce silicone infrastructure or the resources by supporting high-performance on-chip low-power communication.

1.3 Application Areas of ZYNQ-7000-“Data Flow”:

Many data streaming applications like video and multimedia processing or packet switching needs great performance, which can be achieved by mapping the multi-task parallel application on the System on Chip (SoC). A smooth transition of a task from software to hardware implementation demands a unified system for information exchange. Some applications that have relatively modest requirements for multi-processing, and that could be done within the available PL components and resources would be more cost-effectively implemented with the Zynq-7000 platform. Some sets of applications/examples will be demonstrated here which heavily requires data transfer performance analysis among the processors.

1.3.1 Video Processing of Drones

A common use of drones is aerial photography. In recent days, drones are used in individual filming, news, and sports broadcasting, monitoring, security inspection, and agricultural applications, etc. An HW/SW co-design can be used in video and image processing in this application.

However, to achieve professional-quality recording by using drones some parallel combinations of tasks are needed. Fast frame rates, integration of the 4K or higher resolution, and (HDR) video, may result in a huge quantity of the data to be stored.

Video streaming from drones brings new challenges, and current research areas usually involve some compression methods. These methods are appropriate for wide mobility cameras, and drone-based cameras for sports and other live broadcasts. Also, this type of advanced camera technology includes the system to monitor and detect an exact location of an object in the video. All of these tasks regarding video processing involves complicated processing. A software program running on the Zynq SoC-7000 application processing unit (APU), preferably with the advantage of hardware acceleration in the Programmable Logic(PL), would be an optimal solution in this application. For this reason, the data transfer needs to be faster and the suitable interface must be chosen as per the application [33].

1.3.2 LiDAR in Autonomous Vehicle

LiDAR (Light Detection And Ranging), which uses radar-like concepts of laser light, is fast to emerge as an impressive technology particularly for autonomous vehicles. Even though LiDAR systems are currently significantly large and costly, there is an increasing potential to develop lower-cost and relatively portable solutions that are suitable for mass-market vehicles.

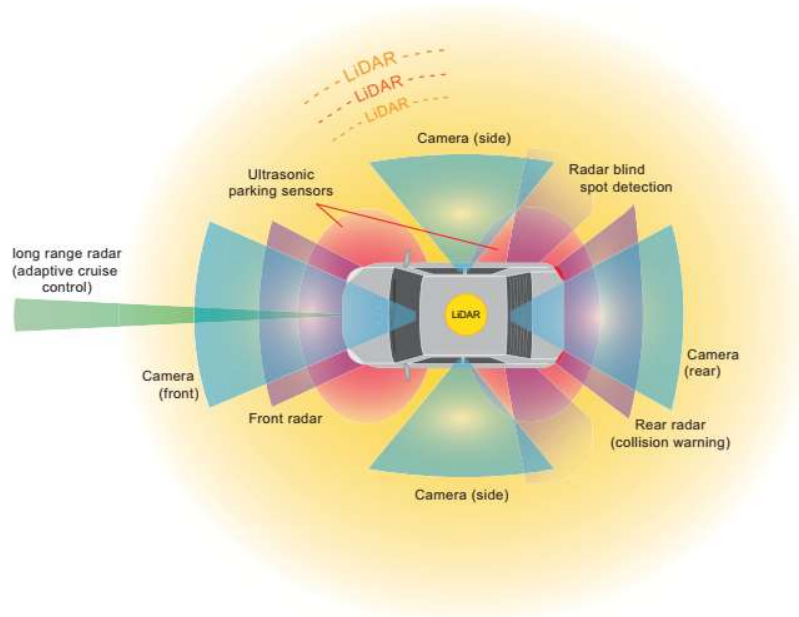


Figure 1-2: LiDAR in Autonomous Vehicle[33]

Figure 1-2[33] shows a typical representation of an autonomous vehicle where different types of sensors are placed around the vehicle. For example, adaptive cruise control, blind-spot detection, etc. Front, rear, and sided cameras can be used to capture visual data elements from the image to gather the sensor information. In this case, LiDAR is an important choice as it can deliver greater distance precision than radar at medium distances (10-100 meters), including complete 360-degree coverage, which is appropriate for autonomous vehicles.[33]

To determine the vehicle's condition, the significant amount of data produced in the sensor systems must be processed, merged, and evaluated. This implies enormous signal processing on sensor nodes input. Also, accurate syncing and further processing to incorporate the data helps to interpret hazards and take immediate measures accurately. All this process (including both PL and PS based components in Zynq-7000) should occur simultaneously as quickly as possible,

considering the high speed at which vehicles can drive and the reaction times required to prevent collisions.

1.4 Thesis Outline

For this thesis, the outline is divided into five chapters. Chapter 2 contains the background or related work on the mentioned topic. At first, a brief overview of the background work related to Hardware-Software Co-design is discussed. Later some related work with ZYNQ APSoC is provided. The present scenario of the available related platform is also discussed. Afterward, the applications of Zynq both in research and the present application has been discussed.

Chapter 3 discusses the Zynq platform, its architecture, and its specific features regarding communication between the processors and operating system descriptions. This section focuses on the PL & PS part interface types and how each port can be used to transfer data from PL to PS and the corresponding IP cores to establish the communication. This chapter also focuses on the software platform that can be used for designing the hardware and the software development tool for PS part considering the Standalone Bare-Metal operating system and describes how each of these interfaces is dependent on each other. This portion also describes the methodology to be followed to implement the design and the analysis.

Chapter 4 describes the design and implementation part using the Vivado software platform and software development kit (SDK). Vivado has been used for designing IP cores with Verilog RTL based designing system. This part also discussed the test bench of the designed IP core. Finally, at the end of this chapter, it discusses the different IP cores and the corresponding designing techniques and the implementation of hardware design to transfer the data from the custom based IP core located at the FPGA part and transferring a stream data to the DDR memory of the Zynq processing system using AXI interconnect. The implementation of the different port-based design is also explained in this chapter. It also discusses the software part implementation using the Software Development Kit (Xilinx SDK) tool. A complete flow diagram of the data/packet moving process is also described in this section. It also discusses the Standalone Bare metal operating system implementation of the processing system.

Chapter 5 describes the performance analysis of the interfaces after the hardware implementation of the design on the Zedboard using JTAG port. It also discusses the data collection of the time in different data transferring conditions for different slave port to transfer

the data from PL to PS. Graphical based performance analysis of the AXI interconnection ports is also presented for the different traffic conditions and the packet size.

Chapter six is the conclusion where the summary of the full thesis is discussed along with the future work scope and analysis plan. A complete flow graph of the thesis outline is presented in Figure 1-3.

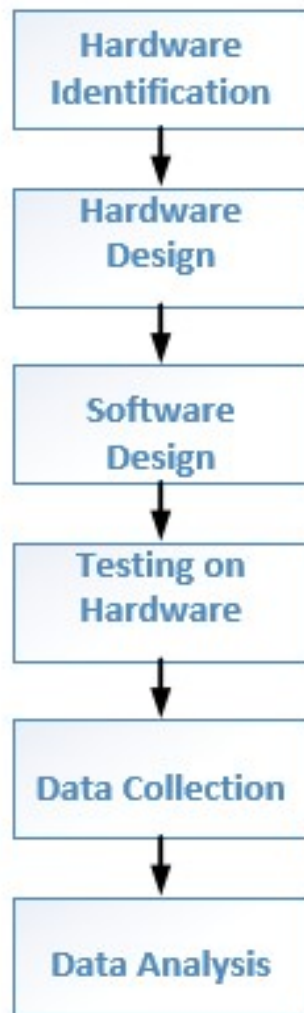


Figure 1-3: A flow graph of thesis outline

2. BACKGROUND AND RELATED WORK

A literature review on the latest cutting-edge method of data transfer for hardware/software Co-Design is the first approach of this work for achieving the goals defined in the previous chapter. There is also a need for a thorough understanding of the Advanced Extensible Interface (AXI) protocol bus and the software and hardware platform.

2.1 Background:

2.1.1 Hardware-Software co-design

The Hardware / Software co-design has been a popular field of research for developers. However, hardware/software (HW / SW) co-design can be defined as a simultaneous development of the system's both hardware and software sides. The HW parts should be run in FPGAs or ASICs on an all-inclusive processor while the SW parts are being translated into a low-level programming language.

ASIC- Application-Specific Integrated Circuits

The integrated circuits ASICs are fully configured for a specific purpose. It can be configured in a specific application where high efficiency and lower energy consumption are the key issues. The optimization process can be controlled by the engineers due to its reconfigurability for a specific application. This implies that for a particular application it produces very high output and low power consumption. By completely configuring the ASICs, engineers can save on additional resources and reduce large-volume costs dramatically.

FPGA- Field Programmable Gate Arrays

As a configurable computing platform, the Field Programmable Gate Arrays (FPGA) has become especially important for high-performance computing applications. It can typically improve performance considerably over the conventional computing platforms while reducing energy consumption considerably. Hardware designs typically take a bit longer time equivalent to software designs. A wide variety of resources are available at FPGA such as the Flip-flop (FF),

LUT, Optical Signal Processing and RAM Block (BRAM). FFs are small gate elements that can store a data bit between cycles. For each particular set of inputs, LUT has an N-bit Table of preset responses. However, DSPs are a block of interconnected computational units like the adder, subtractor, and multiplier. BRAM is a single-port or dual-port RAM block very similar to the fabric of the FPGA.

Zynq-7000 SoC

The Xilinx Zynq-7000 SoC is a group of FPGA and CPU chips integrated into the same chip, which allows effective and quick interaction during software acceleration. Software Acceleration is the method of maximizing system function and delegating performance-critical functions for specialized external hardware so that it can reduce program execution time. The dual ARM Cortex-A9 CPU and two Neon co-processors are used on every 7000 chip array. Each of the dual processors has independently owned high speed, low-power cores, and L1 level and 32 KB data cache. Additionally, 512 KB of L2 cache is shared. This also allows FPGA and CPU to share external memory much greater than the usable internal memory by adopting the DDR3 storage interface. Zynq SoC has a Processing System (PS) and a Programmable Logic (PL) in its internal structure. In the PS part, the application processing unit (APU) is located.

2.2 Related Works with Hardware/Software Co-Design:

Co-designing the hardware/software refers to the simultaneous design of both software and hardware in a system. The software is executed in processing elements such as Central Processing Units (CPU), and Digital Signal Processors (DSP). Application-specific Integration Circuits (ASICs) or FPGAs typically implement hardware logic. Since software and hardware systems have different characteristics, there is potential for a hybrid system to utilize the best of both worlds. The publication [11] analyzes the key properties of co-designing hardware/software: teamwork, flexibility, precision, and complexity.

Hardware/software co-design has evolved considerably since it was first used in electrical systems. Co-design 's earliest use case appeared in the 1980s [12]. Partitioning was the key problem to be addressed during the first generation of co-design. Two approaches have been proposed: (a) starting with pure software and then migrating software functions to hardware; [13],

and (b) starting with a hardware-only system and ending with a codesigned system[14]. Multicore and multiprocessor were used in the second generation, and thus multi-threads were used instead of a single thread. Thread planning is one of the biggest challenges in this approach. Besides, the interface and communication between hardware /software are important because they have a drastic impact on device efficiency and design space [15]. According to [11], the co-design is now in its third generation, where it shortens the time-to-market cycles by optimizing the flow of hardware/software development. Also, different languages for the hardware /software co-design are needed. The language for both hardware and software development should be appropriate as per their platform. Neither Hardware Description Languages (HDLs) nor C/C++ seems to be able to replace the other.

The major difference in hardware/software codesign is described in [12] as compared to pure-software design. They are, 1) Allocation: there are plenty of options for SoC architecture design and designers need to pick suitable resources for their systems among different processors, ASICs, FPGAs, DSPs, etc. 2) Binding: Developers having different resources in SoC(s), have to link applications, tasks or variables to similar resources; partitioning, stated in the first generation, was one kind of binding. In [16] it was proved that the binding mechanism is an NP-complete problem, so finding an optimal solution is time-consuming. 3) Scheduling: several resources are shared (such as processors, memory, and communication bus, etc.) so, real-time analysis is required when complex co-design is being implemented.

Additional factors today influence the development of hardware/software co-design [11]. Heterogeneous SoCs is a new phenomenon where many more resources can be combined into a single chip, e.g. one multi-billion transistor chip includes multiprocessors, DSPs, FPGA, IP, memory, and peripherals. First, the complexity of SoC(s) systems is highly increasing, as many SoCs need to work together. Last but not least, as systems become more complex, it is very common for different subsystems to be introduced by different suppliers, so all entities must share shared standards.

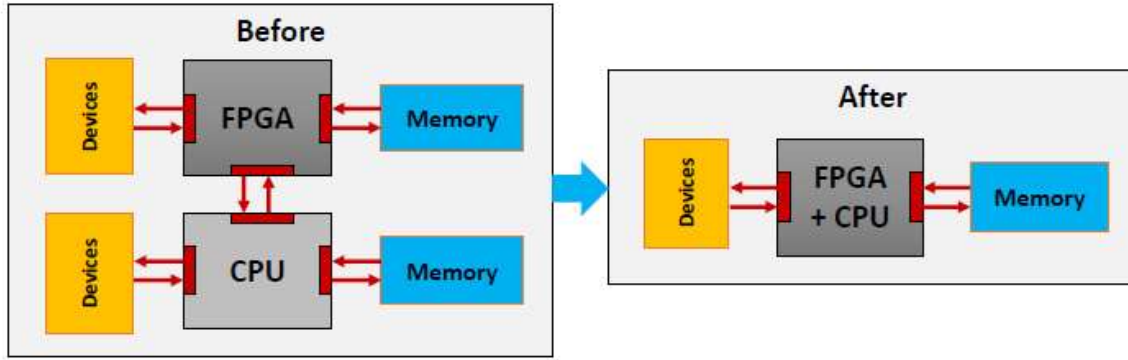


Figure 2-1: FPGA and CPU Integration[11]

2.3 Related Works with ZYNQ 7000:

The Zynq architecture and its flexibility of communication have attracted many researchers to work on it. Particularly in this thesis, all the implementation is based on the Zedboard, which features an XC7Z020 Zynq device. Zedboard is a joint venture between Xilinx, Avnet (the distributor), and Digilent (the board manufacturer) [2]. The books [1] & [2] mainly focus on the Zynq platform and its architecture on different topics.

A paper for analyzing the impact of hardware accelerator data transfer granularity on the performance of a typical embedded system is presented in [3]. An insight into maintaining coherency between CPU caches and accelerator data in a multi-core embedded system is discussed in [4]. The same research paper emphasizes on possible hardware architectures and related software solutions to reduce the problem. The research [4] concludes that the optimal solution is more dependent on the type of application. It also discusses the solutions at the architecture level but it doesn't provide any performance comparisons on the mentioned topic. In [5] the author mainly explores the performance of HP ports under different traffic conditions. However, the study [5] doesn't explore the performance of the ACP and GP slave ports. There are several publications on the implementation of image processing. In [6] the author uses the ZYNQ architecture for image restoring contrast by defogging. This type of application is becoming more popular in automobile industries. Hence, application development using such SOC's is developing rapidly. Another research work discusses the driver awareness monitoring system [8]. A car simulator has been used to prove their method. A project of Microsoft's Catapult was explored in [9] which is mostly concentrated on the acceleration of the Bing Search Engine data center. The project combines the

server rack computers and FPGAs used in achieving a significant improvement in the ranking throughput of each server. This project did not use the Zynq technology.

In this research paper, a custom design is implemented for the data transfer from PL and PS DDR memory. AXI interface is used for the internal data transfer which is on-chip. On the contrary, data transfer between two devices with measurable distance has some critical challenges when the link operates at high frequency, and also the communication becomes hard to maintain. Hence, efficient on-chip communication between two different types of processors was an important topic to ensure the optimal data flow. The performance analysis between the available port to transfer the data from the PL to the PS portion of the Zynq was analyzed in different conditions and a graphical representation of the performance analysis is also discussed.

2.4 Some More Work On A Different Platform

Diverse research on the use of reconfigurable architectures in hardware-software co-design applications has been shown in the industry and academia over the last few years. Studies and experiments on different platforms were also performed. The following are some of these works.

2.4.1 HW/SW Co-design for Exascale System

The study was conducted by the “Lawrence Berkeley National Laboratory” in a joint project involving three laboratories. The experiment was performed on Codesign for Exascale(CoDEx), a robust co-design platform for hardware-software, which provides applications and algorithm researchers with an unparalleled opportunity. This paper discussed the comparative analysis and the experimental results of the “Green Wave” co-design process and the conventional HPC system (Intel Nehalem and Nvidia) in a project to develop climate modeling and seismic imaging applications. The result shows a significant benefit in using co-design applications. The results indicate that the co-designed device provides 5 to 7.6 times higher performance compared to a traditional HPC design for similar constraints lithographic scale, on-chip region, and memory interface technology. [22]

2.4.2 Coarse-Grained Reconfigurable Accelerators (CGRA Architecture)

Several studies were performed recently on the CGRA architecture. In one recent study, domain-specific accelerators (DSAs) has been implemented on a CGRA platform. A new approach for assembling and characterizing the reconfigurable accelerators, based on data flow was implemented in this report. The concepts of functional programming, which can resolve the challenges of the efficient design of CGRA, have been explored. The comparative resource utilization analysis with the other state-of-the-art high-level synthesis(HLSs) is the main advantage of the proposed approach.[23]

In another work [24], the implementation was in the same CGRA based platform. The work described the design and implementation of a “C-programmable hybrid coarse-grained array, single-instruction, multiple-data (CGA-SIMD)” based SDR accelerator. It also analyzes the performance of the design for the SDR based baseband signal processing and also in terms of power efficiency.

2.4.3 Different Embedded System Platform for Co-Design Applications

There are a lot of works with embedded system Co-designing applications. One of the research papers [25] discusses the rapid prototyping in a real-time embedded system platform. A design cycle has been proposed for this framework. The main target of this research is to have an integrated framework Co-design and Rapid-Prototyping System for Applications with Real-Time Constraints. The design flow includes implementation, specification-synthesis, system-synthesis, and performance analysis.

Another work focuses on the unified designing approach for the HW/SW components in an embedded system [26]. However, the goal of this work is mainly to reduce the gap between hardware and the software in the design method. This was done by introducing the object-oriented and aspect-oriented programming, which provides a unified description for an embedded system. The purpose of this work is to explain how the components and element unified design weaving processes can only be applied using the standard interface and its metaprogramming. Hence, the separation of the hardware/software implementation in unified implementation takes place directly via changing at the language level, thus enabling integration with various -related HLS software and flow architecture.

The status and evolution in the field of co-designing of an embedded system has been discussed in [27],[28]. These review papers discuss the major processes in the field of computer-aided embedded systems. In the current system analysis, co-simulation of HW/SW is the main requirement.[27]

However, embedded systems face special problems for the evolution of systems. They are embedded in a dynamic environment and they have to interact with the evolving processes of different organizations. To address these problems the research and the industrial practices need to improve.[28]

2.4.4 FPGA and other SoC Platforms

For low power and distributed applications, FPGA and SoC platforms are widely used for the research and industry sectors. In one of recent the research works,[29] an algorithm has been proposed to determine and manage the problems of resource management/ allocation and task assignment. To assess the timing of all assignments and coordination events in the system, a priority task scheduling was also developed. Another research work represents[30] an experimental analysis and of an SoC platform. This “Dalton Project” shows that platform tuning(parameterization of the IP and IC) can increase the performance of an SoC-based embedded platform.

However, reconfigurable architecture and the computing system has brought immense performance increment in terms of power and resource-based efficiency[31]. Reconfigurable Computing has established groundbreaking accelerator architectures, hardware, and RTR models and support for the past 25 years. A timeline of that evolution can be illustrated in figure 2-2 [31].

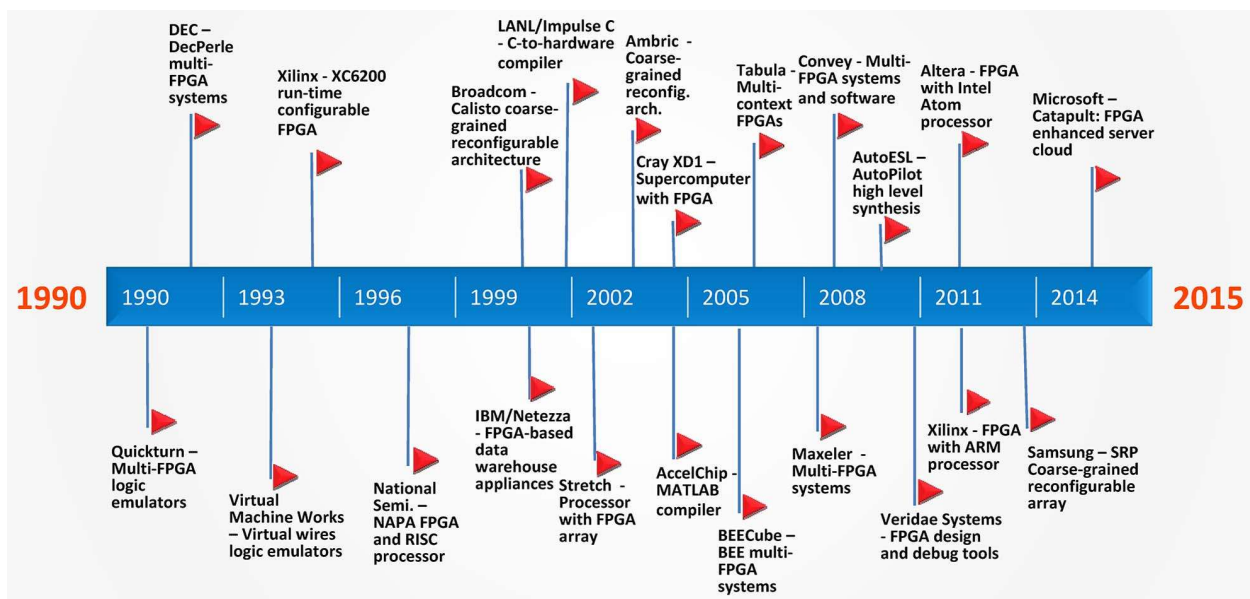


Figure 2-2: A timeline of reconfigurable computing system evolution[31]

3. HARDWARE-SOFTWARE PLATFORM AND METHODOLOGY

In this chapter, we introduce the hardware and software platforms that have been used in the course of this work. First, we describe the Xilinx Zynq-7000 device architecture and the AXI protocol, which manages its internal communication. Then, the software workflow is described that is used in this thesis. In the last section, the methodology for performance analysis is briefly described.

3.1 ZYNQ-7000 Architecture

The analysis assesses the implemented design on Z-7020, which belongs to the Xilinx ZYNQ-7000 SoC family. Detailed architecture and its features are provided in the following sections.

3.1.1 Overview

Xilinx ZYNQ-7000 SoC has a Processing System (PS) and Programmable Logic (PL), as shown in Figure 3-1[18]. The PS comprises of different working blocks like I / O peripherals, internal and interconnection memory interfaces. PS-PL communication is carried out via high-bandwidth AMBA AXI interfaces (define AMBA).

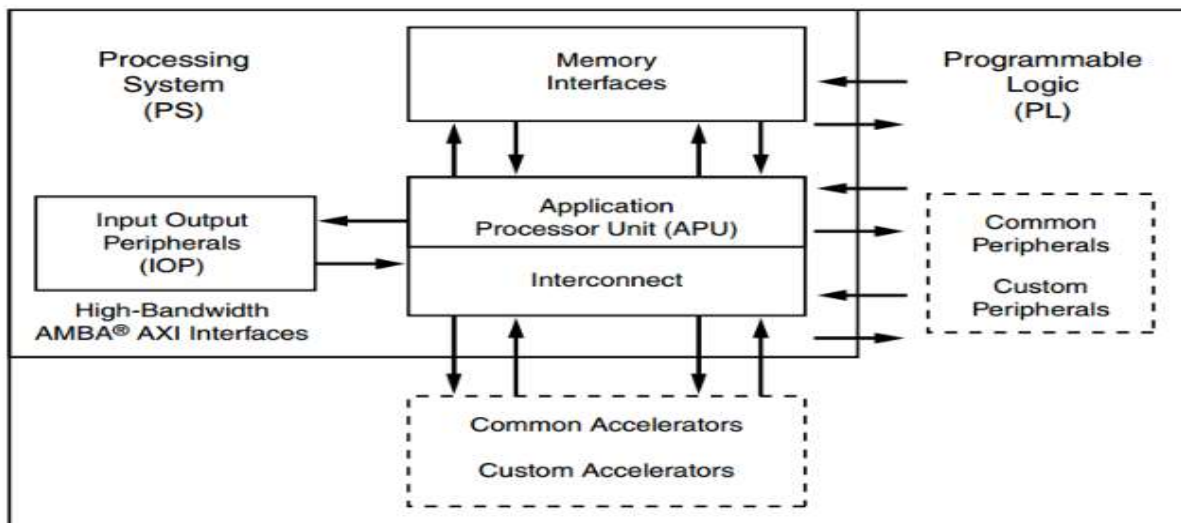


Figure 3-1: Zynq Block Diagram[18]

The ZYNQ framework integrates reconfigured functionality with a PS ASIC interface through this architectural scheme. This is necessary as the PL is capable of performing a parallel data-intensive task, while PS can handle the sequential process. This leads to increased performance overall. The AMBA AXI4 connects a high-bandwidth connection between the PS and PL as well as the connections within the PS. With the AMBA Standard AXI4 Interfaces, a high-bandwidth connection between PS and PL is established. This ensures effective communication with the functional units.

First, the PS is configured. Only after the PS configuration, the hardware design will be loaded into the PL fabric. Once the system starts, a hard-coded boot ROM is performed, which in turn allows PS to load a First Stage Boot Loader (FSBL) from OCM. On the second-stage, the bootloader can be used to load the kernel to DDR memory. A bitstream file can then be used to set up the PL fabric, if necessary.

Zynq-7000 family PL uses Xilinx 7 Series FPGAs such as Z-7020 (which is used in this thesis). The Zynq contains Xilinx-7 series FPGA. An external clock pin or a PS may be used to produce the clock to PL. Three PLLs (phase-locked loops) are used by PS and four input clocks are provided for the PL. Architecture manages clock synchronization between PS and PL. In a wide range of fields, such as industrial motor control, automotive and medical diagnostics, SoC devices Zynq-7000 are used. Figure 3-2 shows the target hardware platform for design and implementation.

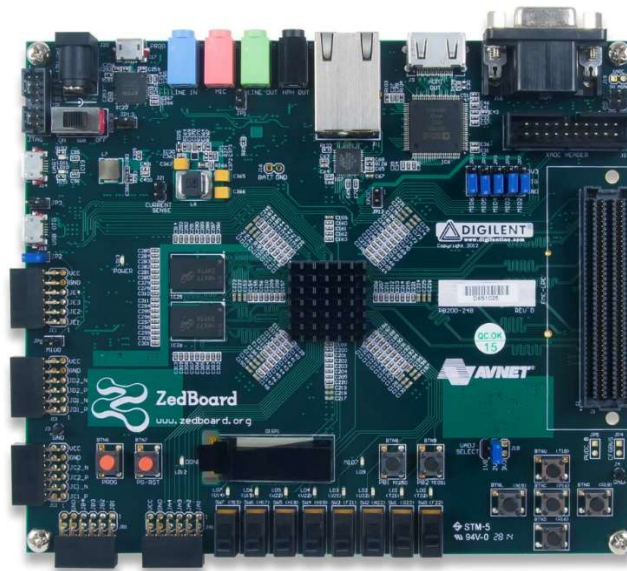


Figure 3-2: Zedboard

3.1.2 ZYNQ-7000 Features

Figure 3-3[17] shows the functional blocks integrated into the ZYNQ-7000. The figure also displays the location of the Processing System (PS) and Programmable Logic (PL) within the same chip.

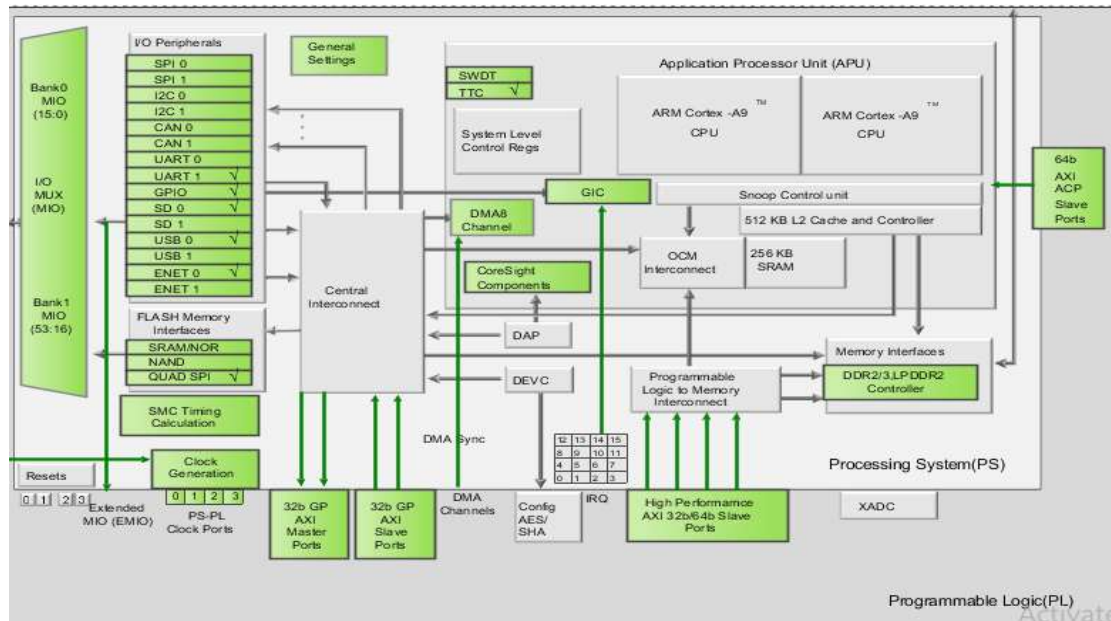


Figure 3-3: Zynq-7000 Architecture[18]

Application Processor Unit (APU) consists of a processing system that incorporates a dual-core ARM-Cortex A9 processor with a 32 kB level 1 cache memory. The Floating Point Unit (FPU) provides the necessary floating operation for applications such as DSP. To improve system functionality by employing data parallelism, the NEON Engine supports “Single Instruction Multiples Data” (SIMD). DMA Controller and Memory Management Unit (MMU) are also components in APU. The processor design implies automatic cache coherence between processor cores [17].

Central interconnection binds the APU to several peripherals. It uses a 64-bit AXI interface [18] based on ARMNIC301 (Network Interface Configuration). The Processing System (PS) has the “Core Sight controller” to debug and track the software design. PS usually communicates with the PL by using two General Purpose ports (GP ports) based on 32 bit of AXI interfaces. To communicate with the functional units in PS, different types of ports are provided in PL. Two 32-bit General Purpose ports are suitable for the transfer of a small amount of data, for example,

control signals. Two master ports and two slave ports are available as General Purpose ports. Four 64-bit high-performance ports provide high-performance memory access to OCM (On-Chip Memory) and DDR memory. In APU, the Snoop Control Unit (SCU) access is provided by the Accelerator Coherence Port (ACP). PL can use OCM and Level-2 cache memory. XADC (Xilinx Analog to Digital Converter) module is included in the PL for "Analog Mixed Signal" (AMS) processing. It offers a dual 12-bit, 1 mega per second ADC, which allows 17 analog external input channels to access[19].

As Figure 3-3[18] shows, both PS and PL devices are mapped to system memory. Zynq has an EMIO (Extended multiplexed I / O) interface for unmapped peripherals in PS, which allows the PL pins in to use those peripherals.

3.1.3 ZYNQ-7000 Communication Interfaces

The Zynq-7000 family uses ARM AMBA4 AXI. An end-to-end protocol between an AXI master interface and an AXI slave interface is established by the AXI Interface. The AMBA4 consists of

three AXI interface types :

- AXI4 Memory-Mapped Interfaces
 - AXI4-Lite Memory Mapped Interface
 - AXI4-Full Memory Mapped Interface
- AXI4 Stream Interfaces

FFFC_0000 to FFFF_FFFF	OCM
FD00_0000 to FFFB_FFFF	Reserved
FC00_0000 to FCFF_FFFF	Quad SPI linear address
F8F0_3000 to FBFF_FFFF	Reserved
F890_0000 to F8F0_2FFF	CPU Private registers
F801_0000 to F88F_FFFF	Reserved
F800_1000 to F880_FFFF	PS System registers,
F800_0C00 to F800_0FFF	Reserved
F800_0000 to F800_0BFF	SLCR Registers
E600_0000 to F7FF_FFFF	Reserved
E100_0000 to E5FF_FFFF	SMC Memory
E030_0000 to E0FF_FFFF	Reserved
E000_0000 to E02F_FFFF	IO Peripherals
C000_0000 to DFFF_FFFF	Reserved
8000_0000 to BFFF_FFFF	PL (MAXI_GP1)
4000_0000 to 7FFF_FFFF	PL (MAXI_GPD)
0010_0000 to 3FFF_FFFF	DDR(address not filtered by SCU)
0004_0000 to 000F_FFFF	DDR(address filtered by SCU)
0000_0000 to 0003_FFFF	OCM

Figure 3-4: System-Level Address map[17]

In terms of performance and functionality, each AXI interface defines a protocol that differs from each other. The developer can choose an acceptable AXI protocol depending on the application. This enhances design flexibility.

AXI4 Memory-Mapped Interfaces

The AXI4 Memory-mapped interfaces have several channels to allow read/write communication through AXI interconnect from the master to the slave. These channels are distinct and include the data signal, address signal, and, control signals as shown in Figure 3-5[17].

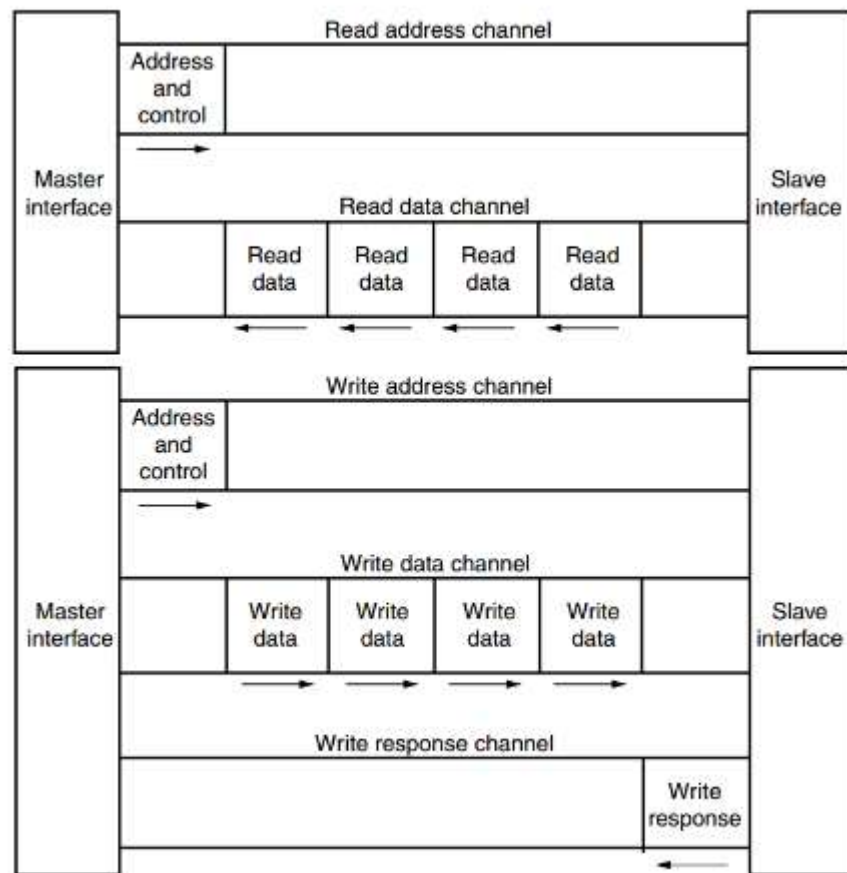


Figure 3-5: Read and Write Channels[17]

Upon transmitting address/control or data signals, each channel employs a two-way VALID/READY handshake process. As shown in figure 3-6[17], only when both VALID and READY

signals are high, the information is transferred. There may be dependence between channel handshake signals that can result in a deadlock. AMBA AXI Protocol Specification thus specifies the dependence rules which must be complied with during the design implementation.

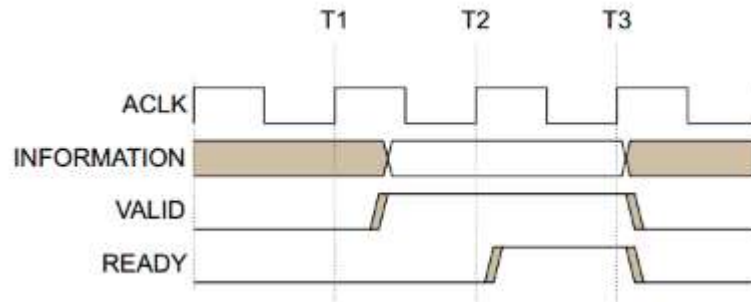


Figure 3-6: Two-way VALID/READY Handshake[17]

Data transmission can be concurrent and bidirectional due to different addresses and data channels to read/write. These channels are used in two kinds of interfaces with memory mapping.

AXI4-Full Interface

AXI4-Full interface applies a point to point burst-based protocol which offers several options for data transfer performance. By placing the first address on the address channel of the burst transfer, AXI Master initiates a data transfer. The slave needs to measure the following transaction addresses based on burst parameters (size and length). Burst size demonstrates the width of a data bit while burst length represents the number of bits in a burst. One of the most important characteristics is mentioned in [20] about the incremental burst size. For the incremental bursts and 1 to 16 bits of wrap bursts, a burst length of 1 to 256 bits is accepted. Support is given for variable data widths between 32 and 256 bits. If required, data can be upsized or downsized. There can be multiple pending addresses. Transactions like non-order transfers and unaligned transfers that provide greater overall throughput are acceptable. Safety features such as read and write access security are also included for the interface. It has an option for adding register slices during pipelines to improve performance.

AXI4-Lite Interface

AXI4-Lite is almost the same as that of AXI4-Full interfaces but does not allow burst transfers. On a different note, it only supports a burst length of 1. The AXI Lite protocol is used by the Processing systems (PS) to set up an IP by mapping the system addresses. This can be used to transfer control signals because only some few clock cycles are needed. This requires the AXI Lite protocol. For example, if AXI Port0 uses, then port 0 address space of the General purpose (GP0) port will be used to interface the IP. This offers a fixed data width of 32-bit or 64-bit. Yet Xilinx IP only has a 32-bit big data bus[17].

AXI4-Stream Interface

For applications that are usually focused on a data-centric and data-flow framework, the protocol of AXI-Stream (for high-speed streaming data) is used. However, the HLS interface has been used to transmit end to end data transmission without the need for an address. This means that data is transferred without addressing from a master node to a slave node. For each AXI4 stream, a one-way channel for handshaking the flow of data is used. This can be shown in figure 3-7[17]. For data flow applications AXI4 Stream IP may be optimized better for performance.

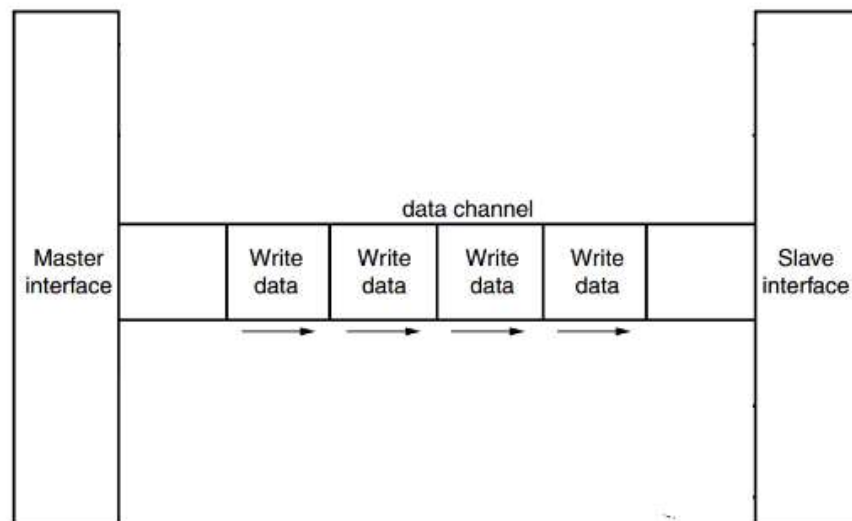


Figure 3-7: AXI4-Stream Transfer with Unidirectional Channel[17]

Our main emphasis for this thesis is the AXI4-stream that is used in our design for the communications system. AXI4-Stream is used in high-speed streaming data. TVALID, TDATA, TREADY, TLAST, and TUSER/ TSTRB are standard AXI4 connectivity signals. The connection TUSER / TSTRB enables additional, fully user-definable information to be transferred. The detail about the AXI stream signals for designing any stream module is discussed in chapter 4. Zynq-7000 has four high-performance (HP) AXI 32/64-bit ports, which are mostly used for reading/writing to the OCM and DDR memory through the AXI4 stream protocol. Certain features of the protocol are [17]:

- **The Stream protocol has an unlimited burst length:** It has different options for choosing the different burst lengths.
- **It has sparse, continuous aligned & unaligned streams:** A sparse stream is the combination of position and data bytes, but usually most of the bytes are data bytes. also continuous aligned stream is defined as the transfer of a number of data bytes that have no position or null bytes in each packet. On the other hand, an unaligned continuous stream may have a number of contiguous bytes at the beginning, end or at beginning and end of a packet.
- **Data transfer features:** It has different data transfer features, for example- interleave, merge, upsize or downsize. Interleaving transmission is the process to interleave transfers from various streams based on transfer types.
- **Ordered transfer :** The stream protocol only allows ordered transfers. The data stream should be ordered in such a way that, the low order bytes of the data stream bus will be the earlier bytes in the stream.

Table 3.1 summarizes the key features and differences of the AXI4 Interfaces.

Table 3.1: AXI4 Feature Availability and IP Replacement

Interface	Features	IP Replacements
AXI4	Memory-mapped address/data interface	PLBv3.4/v4.6, OPB, NPI, XCL
	Data burst support	
AXI4-Lite	Memory-based address/data interface	PLBv4.6, DCR, DRP
	Single data cycle only	
AXI4-Stream	Data-only burst	Local Link, DSP, TRN, FSL

3.1.4 Processing System and Programmable Logic Interfaces

The PS-PL interface includes all the signals to combine the PL and the PS functions. There are two types of PL-PS interfaces:

Functional interfaces: This interface includes AXI interconnect, extended MIO interfaces (EMIO) for most of the I/O peripherals, interrupts, DMA flow control, clocks, and debug interfaces. These signals can be used to connect the user-designed PL IP blocks.

Configuration signal interfaces: This includes the single event upset (SEU), Processor configuration access port (PCAP), configuration status, Program/Done/Init, etc. The configurations signals, which provide PS control, are connected to the fixed logic of the PL configuration block.

For primary data communication between the PS-PL, the following AMBA AXI interfaces are included. These interfaces are under the functional interface family :

- (M_AXI_GP): Two 32-bit AXI master interfaces
- (S_AXI_GP) : Two 32-bit AXI slave interfaces
- High-performance (HP) AXI ports (AXI_HP): Four 64-bit/32-bit configurable buffered AXI slave interfaces. These ports have direct access to DDR memory and OCM.
- AXI slave interface (ACP port) : One 64-bit for coherent access to CPU memory

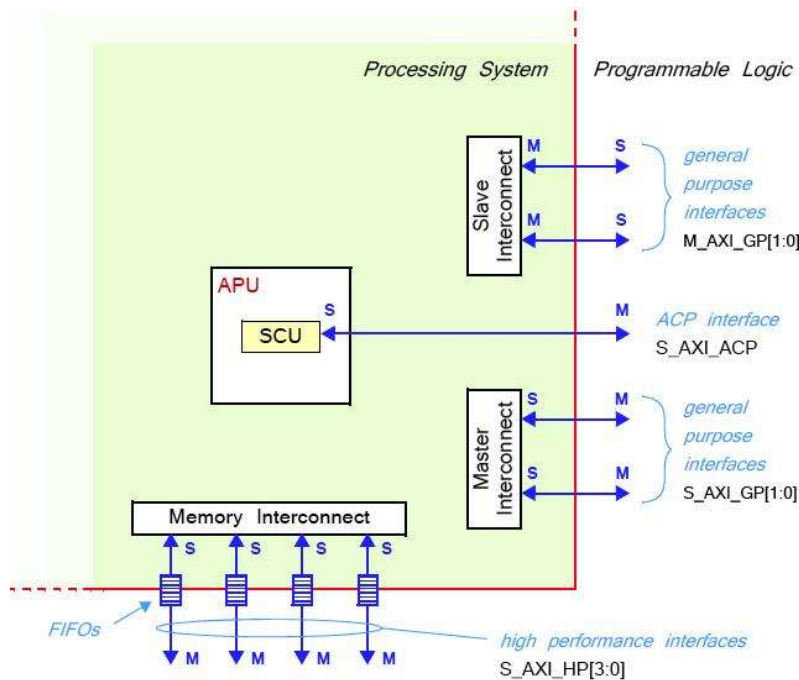


Figure 3-8: Zynq Processing System Architecture[18]

Figure 3-8[18] shows that all of the interfaces are specifically connected to AXI interconnects within the PS, with the only exception being the ACP interface, which is connected directly to the Snoo Control Unit inside the APU.

Inside the processing device, AXI interfaces are used both within the ARM APU (making connections between processing cores and SCU, cache memory, and OCM), and more broadly to link the various interconnections within the PS. These interconnect are internally connected to the central Interconnect as shown in figure 3-8.

To move data from the FPGA part to the Processing system part, the processing system part works as a slave port and the PL part works as a master port. The two GP AXI slave ports (S AXI GP) allow PL AXI master to access the PS memory on the PS subsystem or any of the PS peripherals.

The high-performance AXI ports (HP) and the ACP interface are the two highest-performance interfaces for PS and PL for data transference. The high-performance AXI (AXI_HP) ports provide a high-bandwidth PL slave interface to the OCM and DDR3 memories of the PS. The AXI ACP interface offers a User IP topology similar to the HP ports (AXI HP). Because of its communication within the PS, the ACP varies from the HP output ports. The ACP connects to the L2 CPU cache, allowing ACP transactions to communicate with subsystems in the stack.

3.1.5 Theoretical Data Transfer Throughput of PL-PS Interfaces

Many other documents and descriptions are readily accessible about the use of interfaces as well as how the programmable logic can communicate with the processing system. However, most of the documents didn't fully addressed about the maximum throughput with an experimental analysis using different conditions. In this work a comparative analysis for the three interfaces was presented. However, Xilinx in their document[18] has presented about tthe maximum theoretical (without any overhead protocol) estimated data throughput for a single read/write operations for each of the PL-PS interfaces. Table 3.2 shows the details of that.

Table 3.2: Theoretical Data Transfer Throughput of PL-PS Interfaces

PL-PS Interfaces	Data Bus width(bits)	Maximum Estimated Bandwidth(MB/s)
S_AXI_GP	32	600
S_AXI_HP	32/64	1200
S_AXI_ACP	32/64	1200
EMIO/GPIO	32/64	<25
M_AXI_GP	32	600

3.2 Software Platforms

Designing for the Zynq system requires the use of several different software tools which together form an end-to-end work-flow which will produce a functioning Zynq system.

3.2.1 Vivado Design Suite

Vivado is the tool suite for Xilinx FPGA designs. Vivado is the overall project manager and is used for developing and designing the environment for the configuration of PS, and hardware design for PL. The platform also includes peripheral simulation and logic analyzer. The hardware portion will be primarily designed using Xilinx Vivado software. This program can be configured with the presets of the particular board one is using, which allows it to automatically generate a large number of circuit connections that would otherwise need to be specified manually. It also contains an extensive catalog of hardware IP which can be integrated into one's design. Many of these IP blocks are targeted at signal processing applications such as the Fast Fourier Transform block.

Vivado also contains an extensive set of analysis tools. Hardware expressed using IP blocks or HDL can be simulated and its behavior analysed on the signal level. This can be vital in applications where a single clock cycle delay in a particular operation could cause the system to fail. Additionally Vivado allows for design validation which will quickly pick up any errors or critical warnings in the design before it is processed. This is especially useful as the full bitstream generation flow can take a very long time, over an hour depending on the extent of the design. Vivado also supports on-chip debugging which allows for a live analysis of the signal values while the system is powered on and operating.

The most important functionality that Vivado provides is the ability to take a hardware bitstream that can be loaded onto the board. This is a multistage process that begins with a project specified at the register transfer level (RTL) and ends with a bitstream that describes the FPGA custom hardware. The first of these stages is called synthesis. Synthesis is essentially the process of taking the RTL description which may be a combination of IP blocks and HDL and convert this into a logic gate level representation. Figure 3-9 shows the VIVADO software starting window.

Once the design has been synthesized and a gate-level representation has been generated, it must be converted to be compatible with the FPGA. This stage which seeks to configure the FPGA resources to represent the gate-level logic expression is called implementation [21]. With the design implemented it must be converted to a format that can be loaded onto the board, this process is called bitstream generation.

3.2.2 Software Development Kit (SDK)

SDK is an Eclipse-based Integrated Development Environment (IDE) that has been extensively used to meet the unique needs of the Zynq framework. Using the "Program FPGA" command, the hardware description that has been exported from Vivado will be loaded to the chip via SDK. This hardware description can also be used by the SDK to create a Board Support Package (BSP). The BSP contains drivers that are automatically generated for the hardware that was implemented in the design. This allows the user to communicate with the board in C/C++ code using driver calls.

The SDK's principal function is the development of the software part. When a new application is created, one can choose between a bare metal standalone application or a Linux application. The bare metal application runs directly on the hardware, with no OS layer. That gives an outstanding performance. The Linux application option will create an application. When it builds, it will be cross-compiled using the GNU Linux compiler. This is appropriate for the ARM processor on the chip in use. The cross-compiled application can then be transferred via storage media to the board and executed from the OS running on the board.

Another important aspect of the SDK is its debugging capabilities. Applications can be started on the hardware remotely, and a connection can be maintained that allows SDK to handle the control of the program flow. Breakpoints and operation stepping function just like they would in a conventional IDE but the code is being executed remotely.

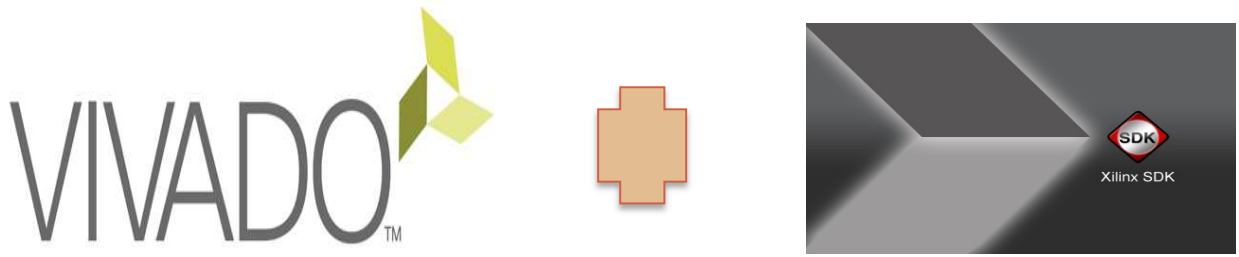


Figure 3-9: Hardware-software design tools

3.3 Methodology

The main target of this thesis is to design a hardware system in order to evaluate the performance of AXI interfaces (ACP, HP and GP). The design is implemented on Zedboard. The AXI Interfaces have to be evaluated under different criteria and conditions to determine the most appropriate interface as per the application. The approach is as follows:

1. Evaluation of three different AXI Interfaces:
 - 1.1. Designing the Hardware in VIVADO for three different AXI4 Interfaces.
 - 1.2. To analyze the performance of three AXI4 interfaces determine several Test cases.
2. Implement the Software algorithm on SDK.
3. Analyze the test result data, and determine the application-oriented usage of each AXI4 interface.
4. The test cases to evaluate the performance of AXI4 interfaces:
 - Data transfer time of write operation DDR memory of PS.
 - Data transfer time for different PL clock frequency.
 - Data transfer time for different data widths and burst transfer.
 - Energy consumption during data transfer.

4. DESIGN AND IMPLEMENTATION

The purpose of this chapter is to present the complete design flow in detail and tools used for implementing the sample data generator. Since the target device is a Xilinx Zynq platform, the Xilinx tool-chain is used. It is composed of Vivado HLS (RTL flow and the HLS), for designing the accelerator and the hardware part and the Xilinx Software Development Kit (SDK) for modifying the original software to work on the accelerated system. The design flow includes hardware design and software design using the tools described in the previous chapter. The hardware design process will be described at first, followed by the software implementation on SDK to run the application on the target board.

4.1 Hardware-Software Design Co-simulation process

A description of the Hardware / Software co-design to be used in this study is shown in figure 4-1. The first step in hw/sw co-design is the high-level specs of the system behavior which includes the behavioral simulation, power, functionality, and other constraints of the expected circuit or design.

The following move is to partition the functionality of device design between the software and hardware. The partitioning of the Hardware / Software is the mechanism that separates the functions into a part of the hardware and software. The partitioning phase gathers the information from the profiling task to make decisions about the instances to map the software and the hardware. Once the instance of hardware and software creation has been established, and the interfaces has been established between them, the next step is the coding & simulation. In this stage, specs are refined, where autonomous system specifications are translated into HW and SW specifications. When the coding & simulation action is over, the subsequent phase is authenticated by the design flow, which collectively simulates both the hardware/software. This is called the co-simulation process.

The co-simulation process verifies whether or not the design objective has been achieved. When the specification is appropriate, the co-simulation ends. The design cycle is repeated until the satisfactory design output is reached; if the design is not satisfactory, the design returns to the HW/SW partitioning step.

If the results of co-simulation are acceptable, then the next task is the simulation-level implementation of both hardware and software parts.

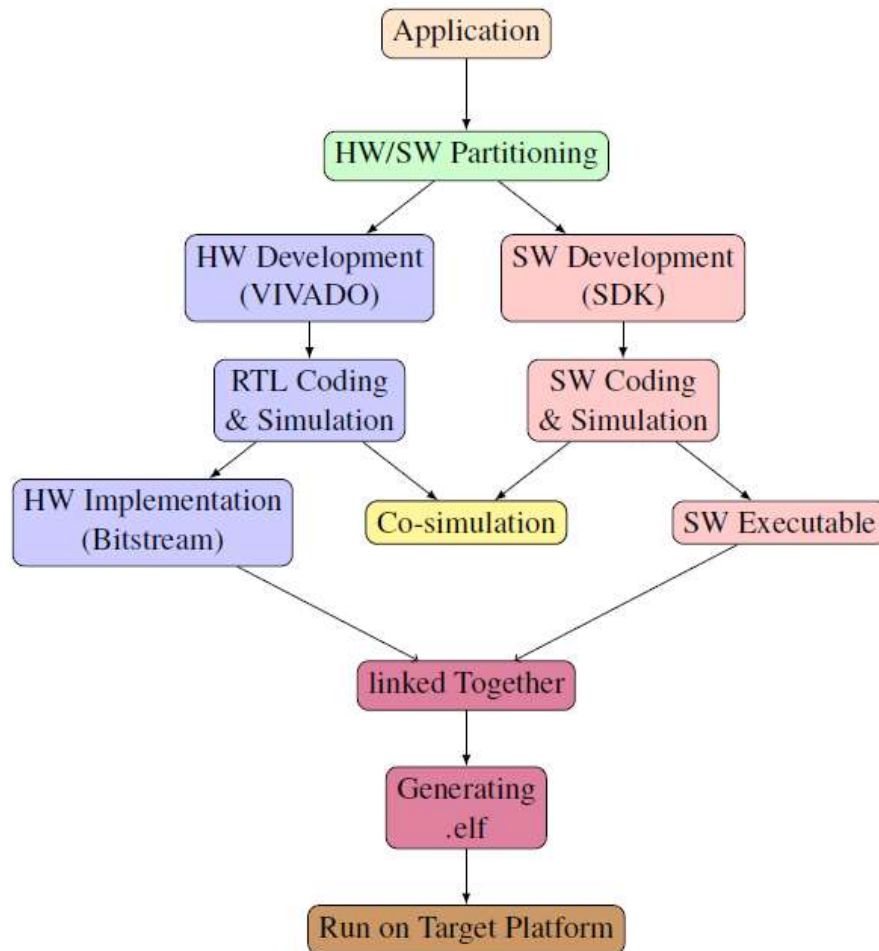


Figure 4-1: Hw/Sw Co-design Flow

Once the partitioning task is finished, the executable program and the bitstream files are combined to create the .elf file and to run it on the hardware platform. The performance of these parameters relies on the level of design constraints and the reduction in design costs. If the design parameters are not fulfilled, then to boost the performance some optimization techniques must be used to obtain the desired performance.

4.2 Hardware Design

This portion describes the hardware design implementation using VIVADO Design Suite software. The main idea is to integrate custom stream IP (Sample Data generator) and through the AXI-Bus interface and DMA engine to establish the communication between the processing system and FPGA. Where, through the custom IP, the FPGA portion will send the data to the DDR memory which is accessible by the ARM processing system. This communication has been established using different AXI slave interfaces.

The hardware design implementation is described in three main sections. Firstly, to start with the proposed design, a custom IP “Sample Data Generator” is designed using the RTL flow. The designed IP is then modified with the Verilog code. Secondly, the designed IP is verified using a Vivado logic flow simulation to test the proper functionality. Thirdly, the “Sample Data generator” block IP will be integrated with the AXI DMA and AXI-Bus interfaces as illustrated above.

4.2.1 Designing “Sample Data Generator” using RTL Flow:

The data generator is a customizable, programmable logic core designed in Verilog language, mostly known as "Hardware Description Language" (HDL). It produces a stream of sample data which the AXI DMA Core finally receives. The clock is created by the PS-configured FPGA common clock. The clock is produced by toggling on each nth edge of the common clock and N is a natural number greater than 1. Finally, the data generator introduces a standard FPGA clock-driven AXI 4-stream protocol. The new value is moved to the stream and to the AXI DMA core, when the counter register is updated.

There are two modules in the sample data generator: the slave port and the internal data generation counter. In Figure 4-2 one can see the a graphical representation of the inputs , outputs, and mutual connections of these modules. At the top level of sample data generator, the AXI4-Stream interface is implemented.. In the designed AXI4 stream module the “En- Enable Signal” is used to enable and disable the module. “AXI_En – AXI Enable” plug is used as an input signal to switch between the S_AXIS and the Frame Size.

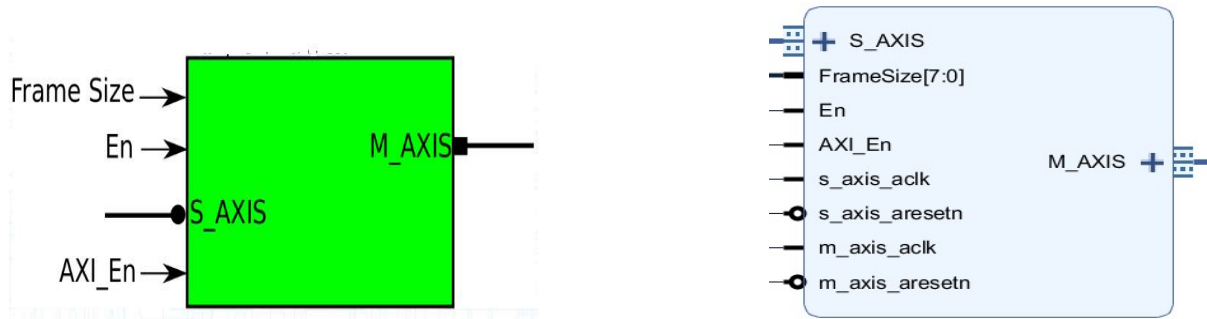


Figure 4-2: The sample data generator AXI stream module

The clock is derived by measuring the rise edges of the general FPGA clock/ACLK as well as by restoring the output as per the user input value. In figure 4-3 and in the entire module in Appendix A, the reset circuit and counter implementation of the sample data generator can be illustrated.

```

1  // sample data generator
2  reg [C_M_AXIS_TDATA_WIDTH-1 :0] counterR ;
3
4  assign M_AXIS_TDATA = counterR ;
5  assign M_AXIS_TSTRB = {C_M_AXIS_TDATA_WIDTH/8 {1'b1}};
6
7  // counterR circuit
8  always @(posedge M_AXIS_ACLK)
9      if ( ! M_AXIS_ARESETN ) begin
10         counterR <= 0 ;
11     end
12     else begin
13         if ( M_AXIS_TVALID && M_AXIS_TREADY )
14             counterR <= counterR +1;
15     end
16
17
18  // circuit to count number of clock cycles after reset
19  reg      sampleGeneratorEnR;
20  reg [7:0] afterResetCycleCounterR;
21
22  always @(posedge M_AXIS_ACLK)
23      if ( ! M_AXIS_ARESETN ) begin
24         sampleGeneratorEnR<= 0;
25         afterResetCycleCounterR <= 0;
26     end
27     else begin
28         afterResetCycleCounterR <= afterResetCycleCounterR +1;
29
30         if (afterResetCycleCounterR == C_M_START_COUNT)
31             sampleGeneratorEnR <= 1;
32     end

```

Figure 4-3: Clock cycle and counter circuit for sample data generator

The clock is being used in the counter data generator so that a counter register is increased each time the clock is changed from 0 to 1. As a result, for an ACLK clock time, the TVALID register is set to 1. The faster ACLK clock is used in the data generation circuit. Figure 4-4 presents the data generation circuit with the implementation of the TVALID and TLAST signal, while *Appendix A* displays the module program as a whole.

```

70      // M_AXIS_TVALID circuit
71      reg          tValidR;
72      assign M_AXIS_TVALID = tValidR;
73
74      always @(posedge M_AXIS_ACLK)
75          if ( ! M_AXIS_ARESETN ) begin
76              tValidR <= 0;
77          end
78          else begin
79              if (! En)
80                  tValidR <= 0;
81              else if ( sampleGeneratorEnR)
82                  tValidR <=1;
83          end
84      // M_AXIS_TLAST circuit
85      reg [31:0] packetCounter;
86      always @(posedge M_AXIS_ACLK)
87          if (! M_AXIS_ARESETN ) begin
88              packetCounter <= 32'hffffffff;
89          end
90          else begin
91              if ( M_AXIS_TVALID && M_AXIS_TREADY ) begin
92                  //if ( packetCounter == (FrameSize-1) )
93                  // packetCounter <= 32'hffffffff;
94                  if ( M_AXIS_TLAST )
95                      packetCounter <= 32'hffffffff;
96                  else
97                      packetCounter <= packetCounter + 1;
98              end
99          end
100
101      assign M_AXIS_TLAST = ( packetCounter == (FrameSize-2) ) ? 1 : 0;

```

Figure 4-4: Data Generation Circuit

4.2.2 AXI4-Stream interface signals

The IP sample data generator is interfaced with 5 signals at AXI4 output.: M_AXI_TREADY, M_AXI_TVALID, M_AXI_TLAST, M_AXI_DATA, and M_AXI_TSTRB. Figure 4-5 illustrates the AXI4-Stream protocol using 5 signals.

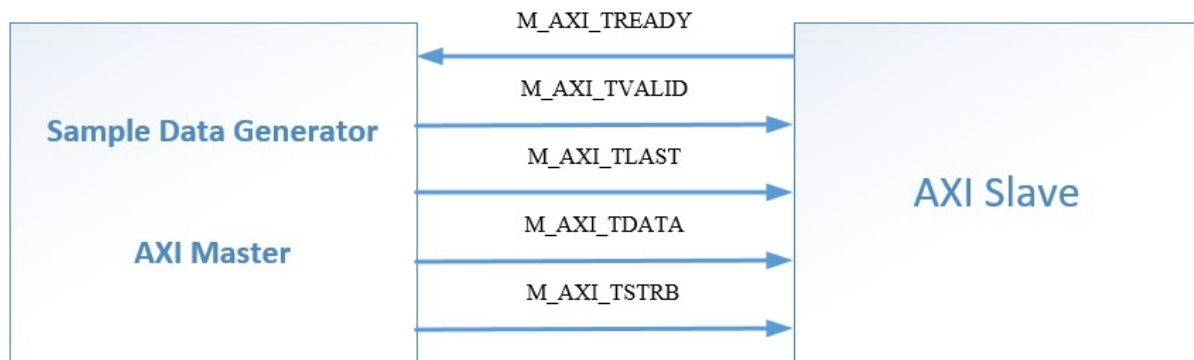


Figure 4-5: AXI4 stream signals

A data transfer for a sample data generator which is an AXI stream module can be seen from figure 4-5. The slave (AXI Slave) must configure the TREADY signal, which indicates that the slave is prepared to accept the data transmission, before transmitting the data. The master will be able to write the stream of data through the TDATA bus after the signal is configured. The master must also configure the TVALID signal, that shows valid data in the stream. Otherwise, the slave will ignore it. After a transfer is complete the master sets the TLAST signal. The TSTRB signal is indicating the amount of valid data bytes in the data stream. For example, in a transfer, the data frame consists of 3 data bits. In this case, while implementing the sample data generator, the 4-bit constant "1111" can be safely written.

4.2.3 Sample Data Generator- Testbench Logic Simulation

A test bench was developed to check proper functionality of the AXIS interface. The behavioral simulation was performed in the Vivado Design Suite, the default design development environment for Xilinx devices. The test bench produces an input of 100 MHz ACLK clock for the generation of a series of sample data frames. figure 4-6 shows the wave-diagram generated by the simulation environment. The testbench was written in Verilog code. The TDATA bus maintains the same frequency of the clock, as seen in the wave diagram created by the Vivado simulator. On the testbench design wrapper level, a customized behavior was written indicating the appearance of the TLAST signal after 16 cycles of the clock. In this wrapper level the block design is wrapped into a vhdl or verilog file so it can be synthesized to generate the output products.

From the below figure it can be observed that, whenever the TLAST signal appears, it shows the end of the last frame of the data. The testbench also shows the proper functionality of the sample data generator. When the “En” signal falls to 0, there is no TDATA or TLAST signal. That means, no stream of data will pass when the “En” signal is not enabled which disables the stream module. These signals are reasonably straightforward. The top-level Verilog-code for the testbench of the sample data generator can be seen in Appendix B. This logic simulation ensures the proper functionality of the designed stream module.

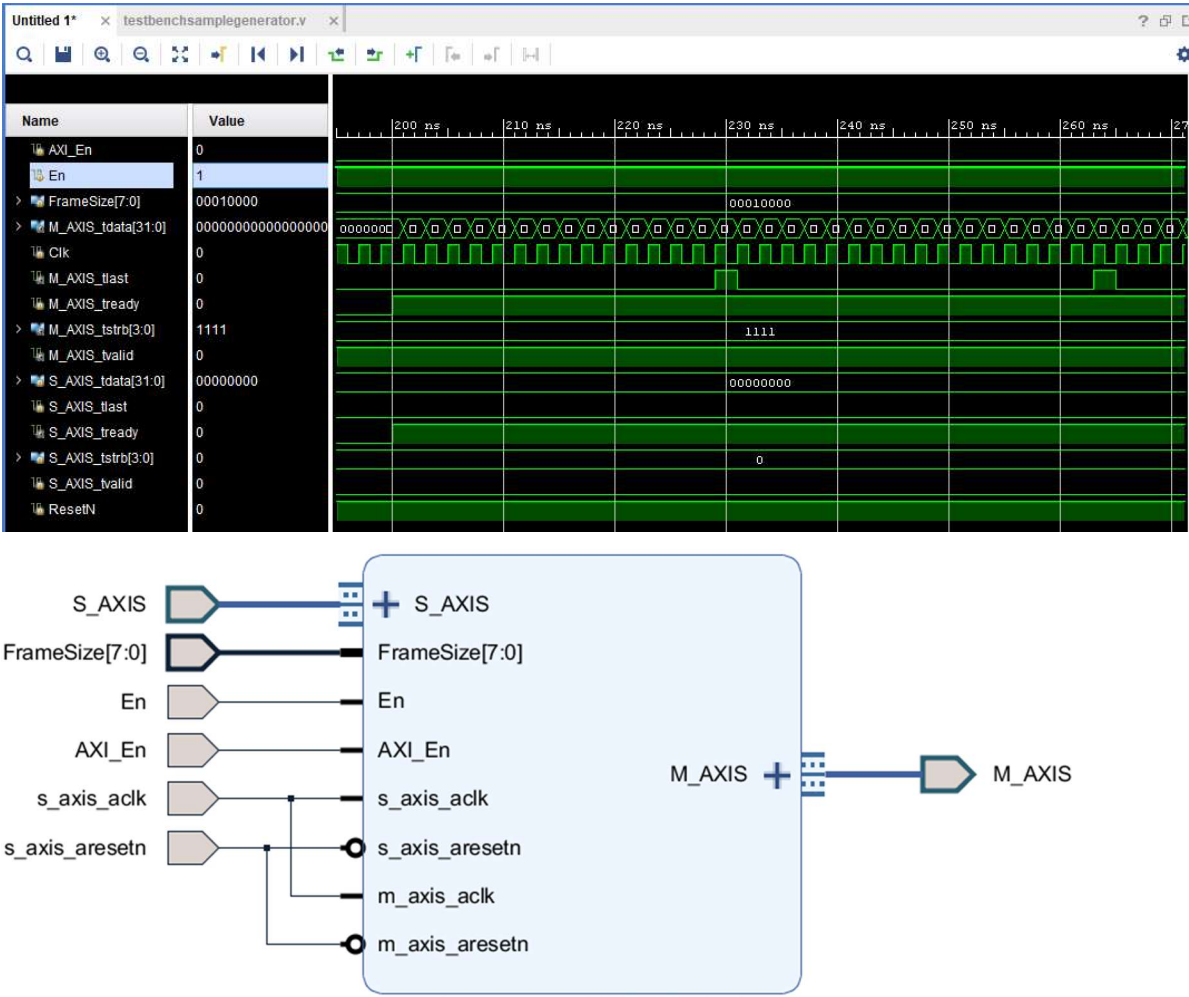


Figure 4-6: The sample data generator test bench simulation showing the logic simulation wave diagram

4.2.4 Hardware Design with VIVADO Design Suite

The hardware design is done in the Vivado design suite. It is used to configure and connect the Zynq Processing System with the other Intellectual Property (IP) cores from the Xilinx Embedded IP library as well as with the custom-designed IP “Sample Data Generator”.

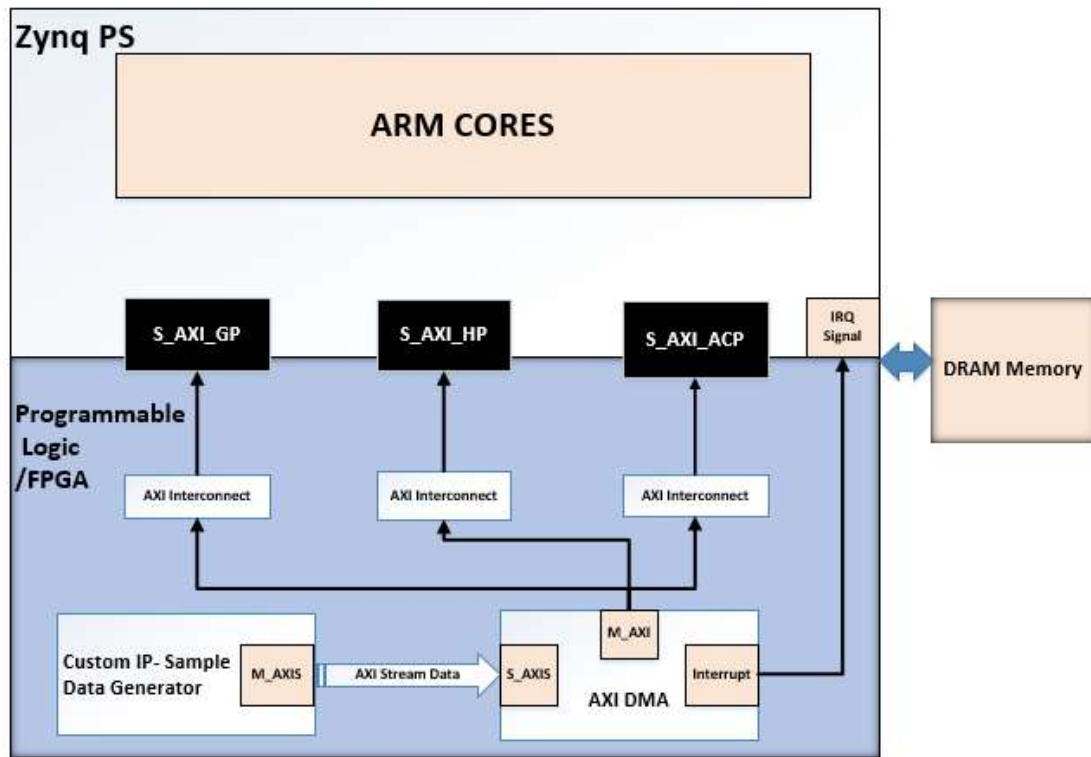


Figure 4-7: Basic Hardware Block Diagram

Figure 4-7 shows the proposed hardware architecture. The custom-designed PL block which is named the “ Sample Data Generator” is responsible for generating a series of data. The series of data is then passed to the PS using AXI DMA. AXI DMA has memory mapped interconnection between the PL and PS. The used memory is specified through mapping. DDR has different pre-allocated ranges of addresses that have to be properly chosen in hardware modules (in the Vivado design suite) and in software modules (in the Xilinx software development kit - SDK). As the data will be passing from the PL side to PS side, the AXI DMA is working as a master port and the available AXI stream interfaces e.g. S_AXI_GP, S_AXI_HP, S_AXI_ACP will be working as a slave port. After the transfer of each packet from the PL side to PS DRAM

memory and interrupt will be initiated from the PS side to start new data transfer. Data transfers between hardware and software are supported by Xilinx IP cores.

From the previous section, when the RTL is successfully verified by the logic simulation testbench verification, the “Sample Data Generator” can be exported in IP format, to be used in Vivado design. This is done by exporting the IP in the repository. The exported design should be copied into the project’s local IP repository, so it appears in the list of available IPs. All the necessary components are added to the design and connected appropriately. The Processing System and the accelerator should share a memory space to store the source and destination grids, which is the DDR memory. The sample data generator can access the DDR through the DMA and AXI slave ports of the Processing System (PS).

Xilinx's programmable logic core, AXI DMA (Direct Memory Access), is implemented in VHDL. It is AXI4- compliant and can be accessed from AMBA bus from the Z-7020 SoC side of the PS (Processing System). It matches with the input and output AXI4 stream defined as stream to memory mapping (S2MM) and memory mapping to stream (MM2S) ports separately. These AXIS interfaces support TDATA buses which are 8, 16, 32, 64, 128, 265, 512, and 1024 bit long. The AXI DMA IP handled all memory transaction address generation, scheduling, and burst formatting. For the other analysis, the burst size was also varied to measure the performance. Most connections are generated automatically via Vivado after initializing the core to the graphical Vivado block design. The sample data generator can now be linked as an AXIS master data source in the design. Figure 4-8 displays the entire Vivado block design of the implemented system. Figure 4-9 shows the mapping of the address for all three interfaces.

The figure demonstrates how the sample generator's AXI4-Stream output M_AXIS links to the AXI DMA core. The GPIO core is used to control the sample data generator via the AMBA bus using the general purpose interfacing port (M_AXI_GP0) is also shown in the design. AXI GPIO cores control and configure the sample data generator. The master port of AXI DMA is connected to the processing system via a high-performance slave interface port (S_AXI_HP0). Similarly, for the ACP ports and the GP ports, the master port of AXI DMA core connects to the (S_AXI_ACP) and (S_AXI_GP0). Before starting the data transfer, the DMA controller should have some specific information which is, the destination address, the source address, & the size of the data that needs to be transferred. The s2mm_introut signal from the AXI DMA is also another important feature of the block diagram. The signal links to an interface (IRQ_F2P), which can

generate system interruptions. DMA begins transferring the sample data to the DDR, while the main data processor performs the other calculations. If the data transfer is completed, the Zynq Processor is sent an interrupt from the AXI DMA to inform the processor that it can complete data transmission.

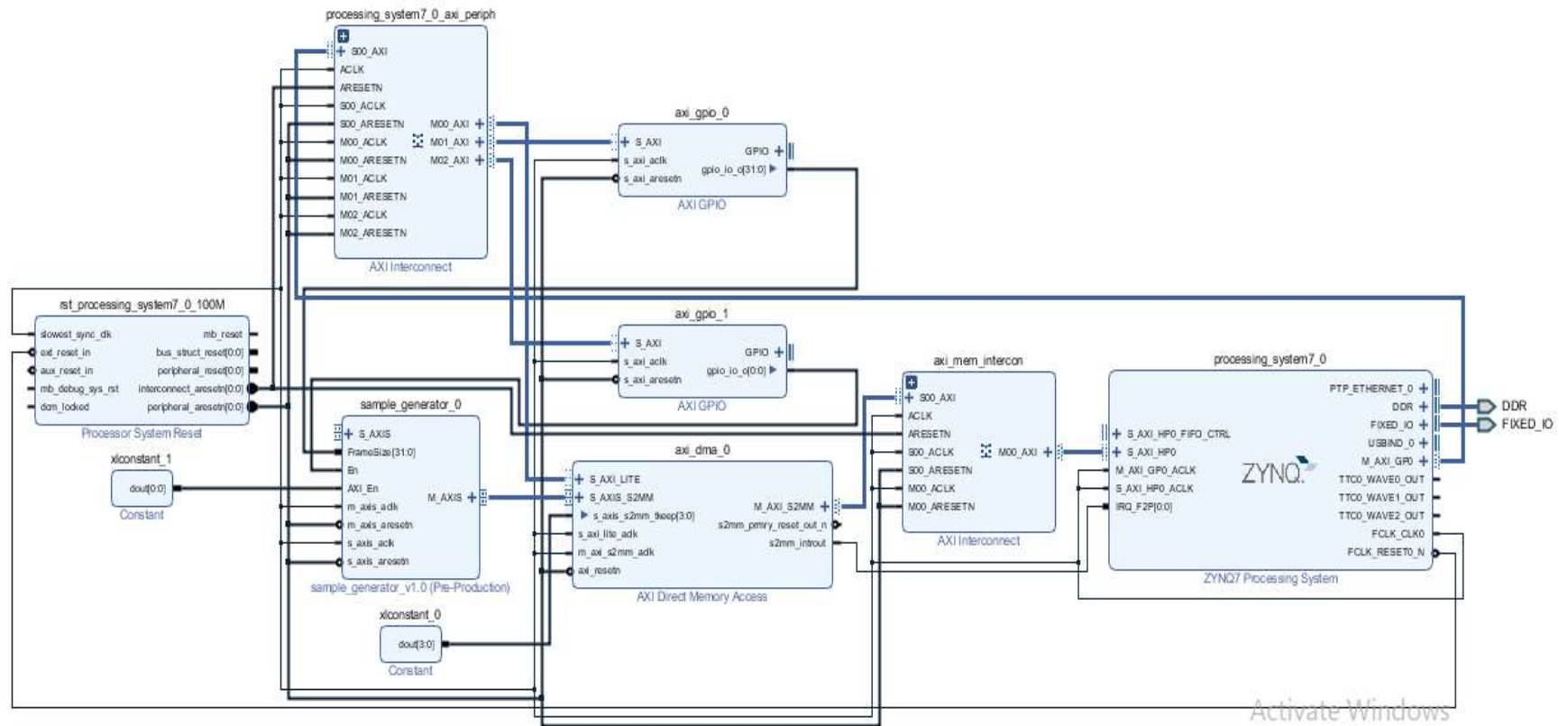


Figure 4-8: VIVADO hardware design with high-performance interfacing port and AXI DMA

DiagramAddress Editor

Q

≡

⬇

⬆

Cell	Slave Interface	Base Name	Offset Address	Range	High Address
axi_dma_0					
Data_S2MM (32 address bits : 4G)					
processing_system7_0	S_AXI_HP0	HP0_DDR_LOWOCM	0x0000_0000	512M	0x1FFF_FFFF
processing_system7_0					
Data (32 address bits : 0x40000000 [1G])					
axi_dma_0	S_AXI_LITE	Reg	0x4040_0000	64K	0x4040_FFFF
axi_gpio_0	S_AXI	Reg	0x4120_0000	64K	0x4120_FFFF
axi_gpio_1	S_AXI	Reg	0x4121_0000	64K	0x4121_FFFF

DiagramAddress Editor

Q

≡

⬇

⬆

Cell	Slave Interface	Base Name	Offset Address	Range	High Address
axi_dma_0					
Data_S2MM (32 address bits : 4G)					
processing_system7_0	S_AXI_ACP	ACP_DDR_LOWOCM	0x0000_0000	512M	0x1FFF_FFFF
processing_system7_0	S_AXI_ACP	ACP_QSPI_LINEAR	0xFC00_0000	16M	0xFCFF_FFFF
Excluded Address Segments (2)					
processing_system7_0	S_AXI_ACP	ACP_IOP	0xE000_0000	4M	0xE03F_FFFF
processing_system7_0	S_AXI_ACP	ACP_M_AXI_GP0	0x4000_0000	1G	0x7FFF_FFFF
processing_system7_0					
Data (32 address bits : 0x40000000 [1G])					
axi_dma_0	S_AXI_LITE	Reg	0x4040_0000	64K	0x4040_FFFF
axi_gpio_0	S_AXI	Reg	0x4120_0000	64K	0x4120_FFFF
axi_gpio_1	S_AXI	Reg	0x4121_0000	64K	0x4121_FFFF

DiagramAddress Editor

Q

≡

⬇

⬆

Cell	Slave Interface	Base Name	Offset Address	Range	High Address
axi_dma_0					
Data_S2MM (32 address bits : 4G)					
processing_system7_0	S_AXI_GP0	GP0_DDR_LOWOCM	0x0000_0000	512M	0x1FFF_FFFF
processing_system7_0	S_AXI_GP0	GP0_QSPI_LINEAR	0xFC00_0000	16M	0xFCFF_FFFF
Excluded Address Segments (2)					
processing_system7_0	S_AXI_GP0	GP0_IOP	0xE000_0000	4M	0xE03F_FFFF
processing_system7_0	S_AXI_GP0	GP0_M_AXI_GP0	0x4000_0000	1G	0x7FFF_FFFF
processing_system7_0					
Data (32 address bits : 0x40000000 [1G])					
axi_dma_0	S_AXI_LITE	Reg	0x4040_0000	64K	0x4040_FFFF
axi_gpio_0	S_AXI	Reg	0x4120_0000	64K	0x4120_FFFF
axi_gpio_1	S_AXI	Reg	0x4121_0000	64K	0x4121_FFFF

Figure 4-9: ZYNQ memory mapping for the custom design interfaces (HP, ACP, and GP ports) in VIVADO

4.3 Software Design

In this thesis, the software design is done using Xilinx SDK. It can be exported to the software development kit (SDK) after the hardware design has been synthesized, implemented, and converted to a bitstream in Vivado. A bare-metal application was used which directly runs on the ARM host without any operating system. A Standalone Board Support Package (BSP) is added to the project to run the application on the target device. It contains all the libraries required to run the bare-metal application along with the configuration headers, which define the peripheral addresses. However, when communicating with the hardware, the processor still needs to know where exactly it should send data to. As the entire device is connected by AXI interfaces when the hardware platform is created, the configured peripherals become memory mapped within the reserved custom logic address space. The required addresses are found within the BSP.

Table 4.1: ZYNQ Memory Map

Start Address	Description
0x0000 0000	External DDR RAM
0x4000 0000	Custom Peripherals
0xE000 0000	Fixed I/O Peripherals
0xF800 0000	Fixed Internal Peripherals
0xFC00 0000	Flash Memory
0xFC00 0000	On-Chip Memory

A peripheral-communicating software program begins by initializing the DMA controller driver. It initializes the DMA interrupts, too. This includes binding a suitable service interrupt routine to be called upon when a specific interrupt is called. When the data transfer to the device is complete, the interrupt routine is called.

Test cases were implemented in C for the evaluation of the mentioned hardware designs. Each interface is tested for PL-initiated write operation to access DDR memory via PS. Testing takes time to complete the operation for a given input data length defined as packet size and packet number. Elapsed time is measured using the library function declared by the header file "xscutimer.h". "XScuTimer_LoadTimer" timer load register will update the timer counter register with the new value. Before starting the data transfer "XScuTimer_Start" register timer will be

started and after the full data transfer is completed “XScuTimer_Stop” will stop the timer. “XScuTimer_GetCounterValue” register returns the current timer counter register value.

The data transfer process needs to complete the following steps to run the algorithm into the ARM host and also to program the FPGA.

- Import the hardware (Bitstream) from the VIVADO.
- Enable the FSBL (First Stage Boot Loader) on ARM host to enable the Bare-metal Application which is a standalone operating system on ARM host.
- The FSBL can be created in SDK and takes the form of an ELF file. It is responsible for the early stages of the boot process including determining which pieces of hardware are to be used and need to be powered on.
- Initialize Level Shifter. This level shifter enables the input and output signals which are routed through voltage level shifters.
- Program FPGA
- Connect the ARM host using the XMD command prompt shell.
- Enable PL clocks and PL-PS interfaces.
- Initialize DMA
- Enable the Sample Data Generator through GPIO
- Set the interrupt system and interrupt handling
- Start the packet/data transfer
- Record the time using the timer after each packet transfer.
- Data transfer time is shown through the UART port where the messages between Zynq and the development computer can be visualized.

A number of data have been passed using the sample data generator for different packet size and numbers. The flow chart of the whole algorithm can be illustrated in figure 4-10.

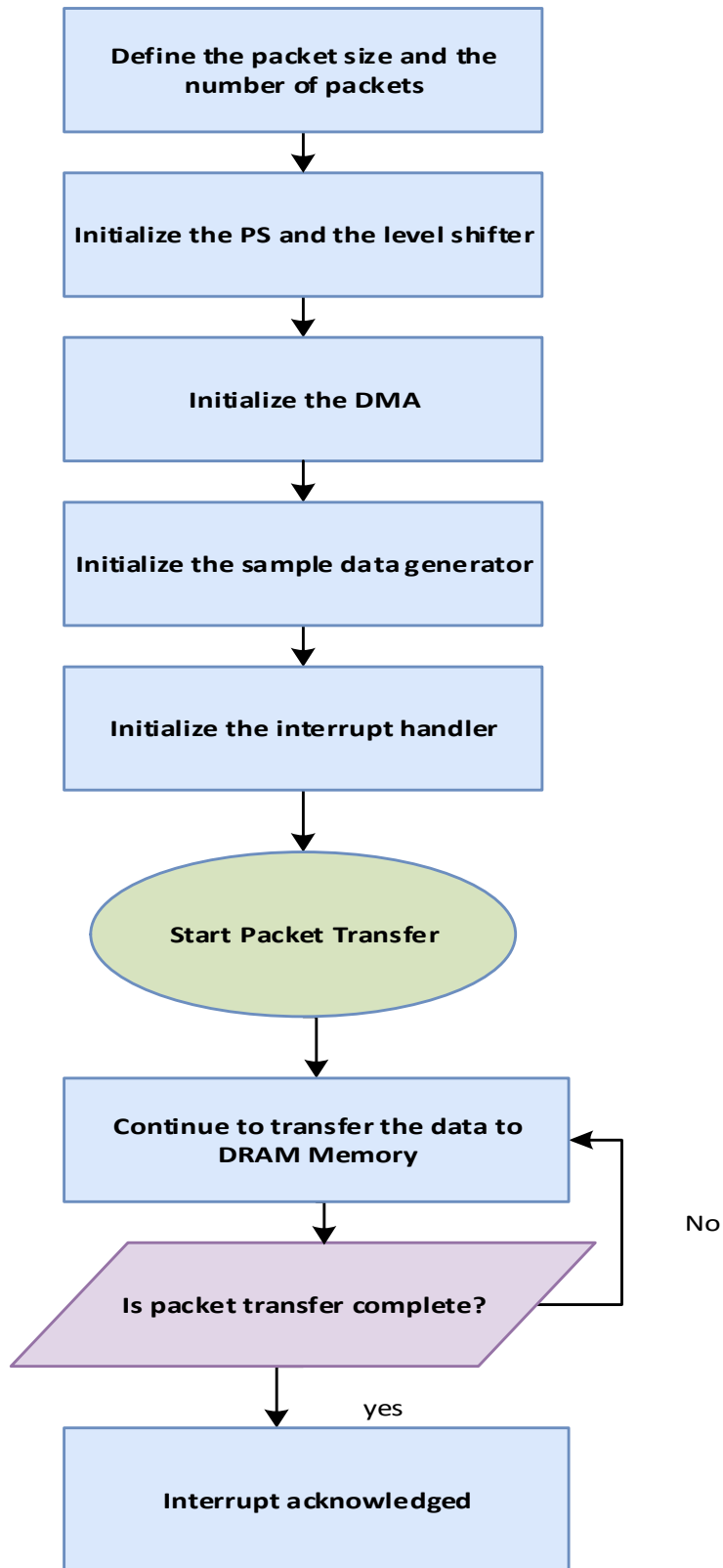


Figure 4-10 : Flow chart for the data transfer from PL to DDR memory

5. EXPERIMENTAL RESULTS AND PERFORMANCE ANALYSIS

In this chapter, two test cases have been discussed to evaluate the performance of the proposed data transfer method from PL to PS DDR memory using the three types of AXI4 ports. Using the first case, we show how the transfer time varies, in terms of the data size for each of the ports. A comparison analysis has been presented graphically. The energy consumption was also measured during data transfer time. The second case was to measure the performance for different traffic conditions and PL clock and burst size for each of the interfaces.

5.1 Experimental Setup

Hardware design is implemented on the Zedboard, featuring a Xilinx Zynq™-7000 All Programmable SoC. Together with the FPGA on the same chip, the ARM Cortex™-A9 provides a powerful and flexible platform for high-performance accelerator application implementation. The OS running on the board is bare-metal without any operating system requirement. The Software development is performed on a host computer, with main memory Intel® Core i5, 2.66GHz, and 4 GB, and Xilinx Architecture Suite 14.2. Power measurements are performed using a multimeter.

The Zedboard features a UART controller and a JTAG to communicate with the host computer. These controllers have a USB interface that allows them to be connected via simple USB cable. The UART is used to transfer messages between the Zynq computer and the development computer. The UART port was configured with a 115200 bps baud rate, 8 bits of data, no parity, and one-stop bit. The host computer runs "Tera Term" to communicate via UART to connect to the COM port. JTAG controller helps host machine to program and debug the Zynq. Figure 5-1 shows the experimental setup for the evaluation with the Tera Term console.

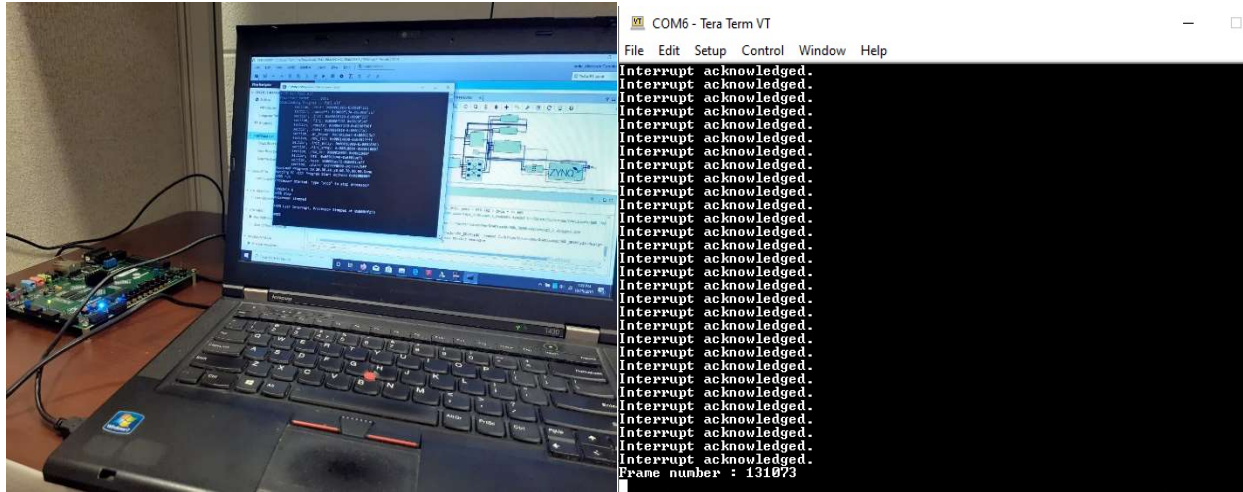


Figure 5-1: Experimental Set up and the Tera Term Serial COM port

5.1.1 Power Measurement in Zedboard

The application code for the software runs on Zynq bare-metal. A current sense resistor which is installed on the Zedboard is used to calculate the power consumed by the Zedboard while executing the designed application. The shunt-resistor is a 10mili-Ohm series resistor with the board's input supply line, and this resistor can be used to measure the entire board's current draw. The current consumption can be used to measure the board's power usage, using the rule of Ohm and the supply voltage. Since the voltage of the power supply is 12V, the board's power consumption is:

$$P = \frac{V_{measured}}{10m\Omega} \times 12V \quad (1)$$

The current sensor shunt resistor is connected in parallel with the header 'J21', which allows easy measurement of the voltage over the shunt resistor. The location of 'J21' is shown in Figure 5-2.

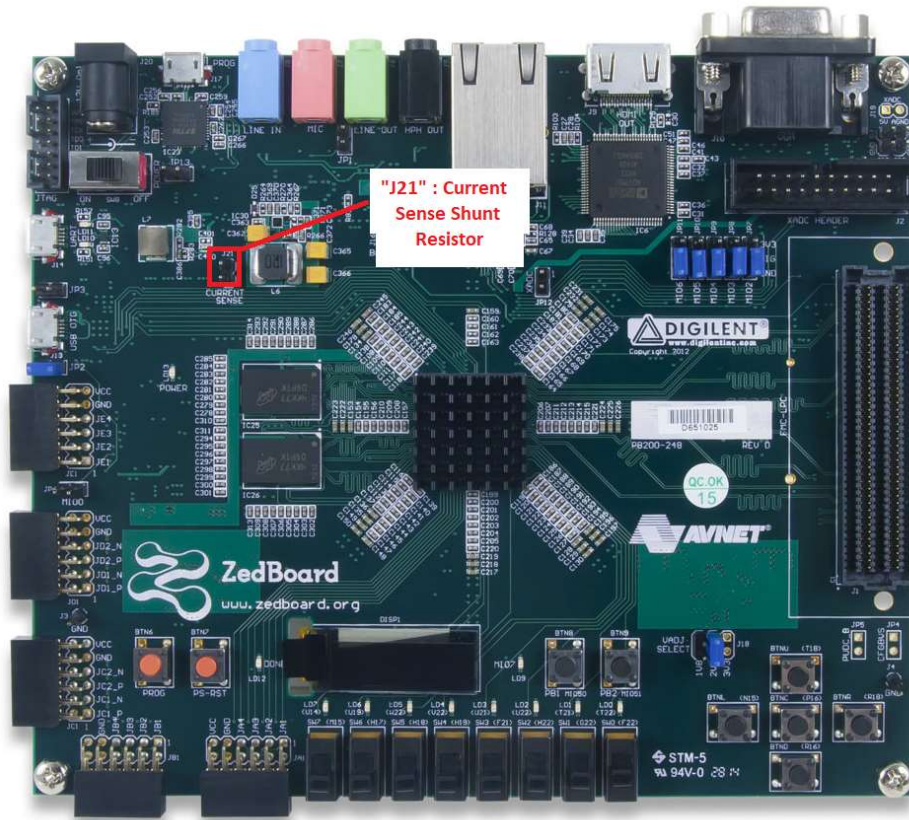


Figure 5-2: Current sensor shunt resistor on Zedboard

5.2 Performance Analysis For Different Data Length

A test case was developed to assess the performance of the three interfaces for different data lengths. In this case, the length of the burst and the frequency of PL were maintained constant. A program running in the bare-metal system inside the ARM cores tested the various interfaces by using the sample generator to pass a different data size to DRAM memory through AXI DMA. The word size for each interface was set to the maximum during the first analysis, e.g, 64-bit for HP and ACP and 32-bit for GP. The FPGA clock was set to 100MHz clock speed.

A data length of 16bytes to 65536bytes has been transferred to the DRAM memory for each of the interfaces. Using the timer, the transfer time has been recorded. The below table shows the relative performance analysis data among the HP, ACP, and GP slave interfaces in transferring the data. Figure 5-3 shows the graphical analysis among them.

Table 5.1: Data Transfer Performance Analysis for GP,1-HP and ACP ports

Packet size(Bytes)	GP-Port		1-HP Port		ACP Port	
	Time (ns)	Speed(MB/s)	Time (ns)	Speed(MB/s)	Time (ns)	Speed(MB/s)
16	2060	7.767	1870	8.56	940	17.02
32	2200	14.545	1920	16.67	1790	17.88
64	2470	25.911	2160	29.63	1930	33.16
128	2650	48.302	2360	54.24	2080	61.54
256	3100	82.581	2610	98.08	2410	106.22
512	3930	130.280	3340	153.29	3050	167.87
1024	5790	176.857	4680	218.80	4360	234.86
2048	9670	211.789	7450	274.90	6930	295.53
4096	17210	238.001	12810	319.75	12090	338.79
8192	32090	255.282	23700	345.65	22270	367.85
16384	61980	264.343	45480	360.25	42710	383.61
32768	121710	269.230	88950	368.39	83700	391.49
65536	241370	271.517	176050	372.26	165640	395.65

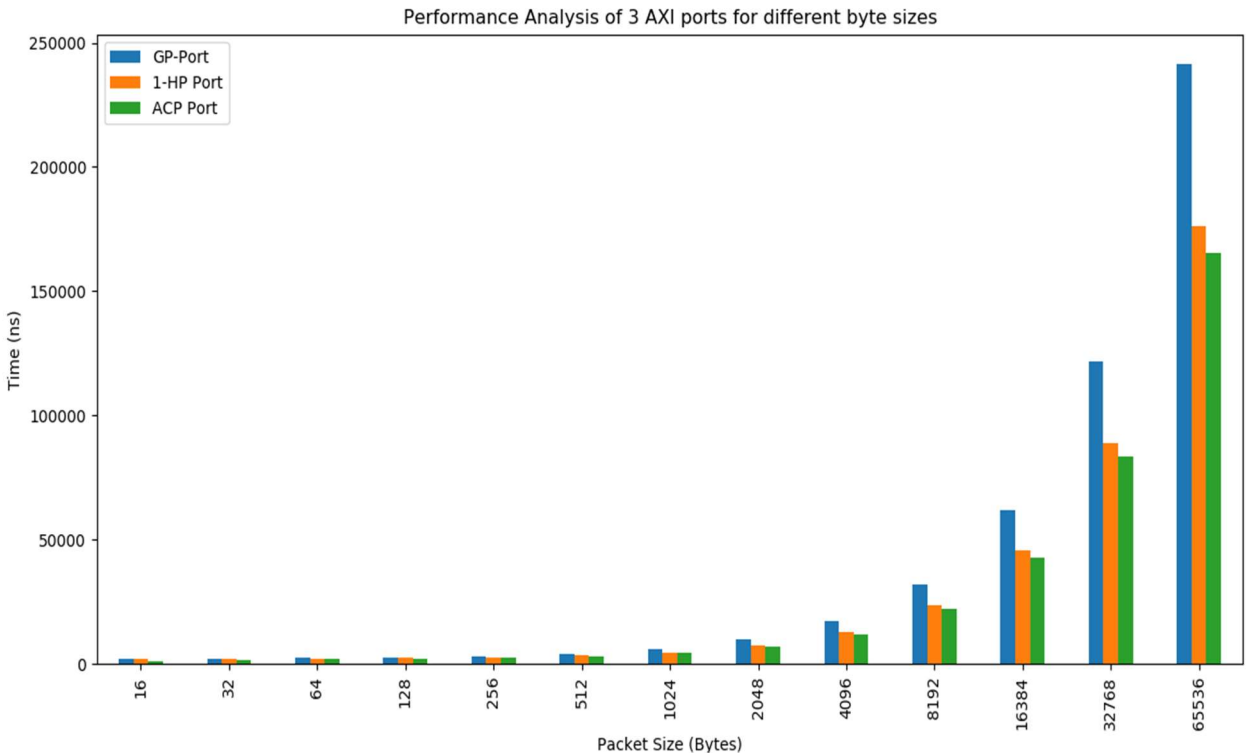


Figure 5-3: Performance Analysis of three AXI ports in transferring data from PL to PS

Figure 5-3 and Table 5.1 shows the data transfer performance of the PS/PL interfaces using AXI-Stream protocols. In figure 5-3, the vertical axis shows the packet transfer time. On the other hand, the horizontal axis shows the packet size in bytes.

The interface with the lowest transfer time is ACP (Green column), achieving highest speed when moving significant amounts of data. The ACP is closely followed by HP port (Orange Column). GP port (blue column) also achieves very high speed but, since it uses a 32-bit data bus, the transfer time achieved is much higher. However, up to 1024 bytes, the data transferring time (from the PL to PS) difference between the three ports is very negligible. High-performance port and the ACP port has shown almost the same performance. ACP port has the lowest data transfer time. The transferring time between HP and ACP port is very small. Transferring data through ACP is on average about 1.15 times faster than a single HP port.

5.3 Energy Consumption

The Zedboard's power consumption was calculated during the hardware accelerator and the software-driven operation for data transfer. The power consumption measured is the difference between the idle /static power consumption and active/dynamic power consumption. The active power consumption is that which will be used for data transfer in all the energy. It's an approximation of the power added to the baseline idle power by a given system configuration. The idle power consumption for all measurements serves as a common reference. Table 5.2 displays the specifics of the measured power consumption by using equation 1. To find the $V_{measured}$ from Zedboard, the "J21" pin serves as the current sensor.

Table 5.2: Power Consumption during data transfer

V_{measured} (Dynamic)	V measured (Idle)	Power (Dynamic) [W]	Power(idle) [W]	Power consumption [W]
2.775mV	2.55mV	3.33	3.18	0.15

From the last column of Table 5.2, it can be observed that the difference in power consumption is very low. Hence, the difference in energy consumption depends heavily on the execution time for the data transfer. Using the execution time for the data transfer of the designed

application the comparison of energy consumption can be calculated. The energy for the transfer time is given by equation 2:

$$E(\text{transfer time}) = P(\text{consumption}) * \text{Time for packet transfer} \quad (2)$$

The energy consumption for the packet transfer is listed in Table 5.3.

Table 5.3: Energy Consumption

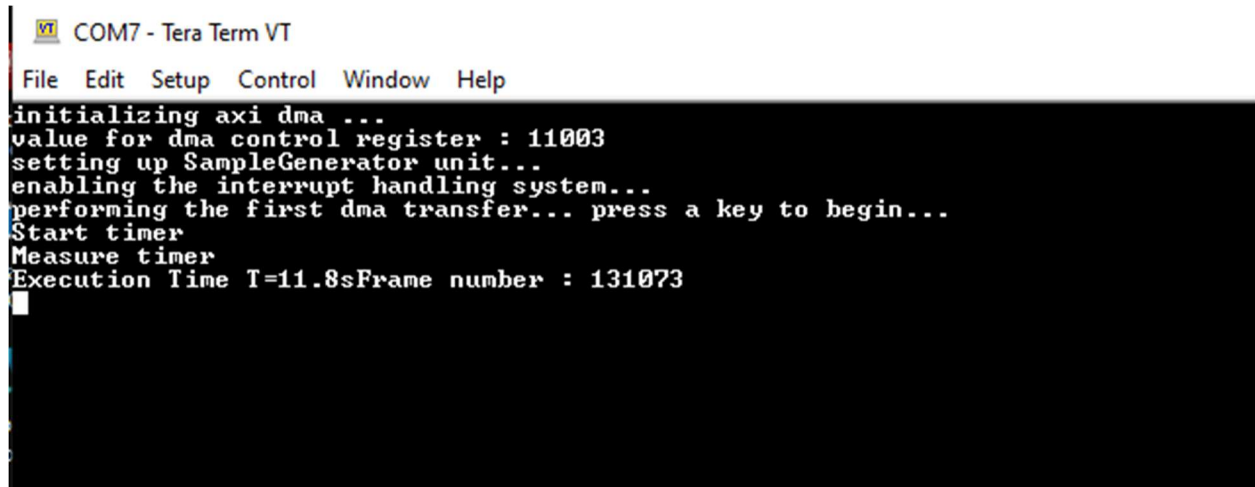
Packet size(Bytes)	Energy (nJ)		
	GP-Port	1-HP Port	ACP Port
16	309	280.5	141
32	330	288	268.5
64	370.5	324	289.5
128	397.5	354	312
256	465	391.5	361.5
512	589.5	501	457.5
1024	868.5	702	654
2048	1450.5	1117.5	1039.5
4096	2581.5	1921.5	1813.5
8192	4813.5	3555	3340.5
16384	9297	6822	6406.5
32768	18256.5	13342.5	12555
65536	36205.5	26407.5	24846

5.4 Performance Analysis For Different Burst Size and PL Clock Frequency

The second case mentioned earlier is to measure the performance for different burst sizes of data and also by varying the PL clock frequency. This condition was implemented for each of the interfaces. A fixed amount of data (4294967296 Bytes) was transferred from PL to the DDR memory using the AXI DMA. In this case, keeping the total data size constant, only the packet size and the number of packets, that is the traffic size was varied to measure the speed of the transferring data.

Three cases were implemented. For the first case, the burst size and the PL clock frequency are 16 bits and 100 MHz respectively. For the second case, it is 256 bits and 100 MHz and for the third one, it was set to a maximum 256 bits and 125 MHz. However, this same condition was applied to test all three interfaces, HP0 (High-Performance Port), ACP (Accelerator Coherency Port), and GP (General Purpose Port). The results can be illustrated in both tabular and graphical

format to analyze the changes due to different traffic conditions. Fig 5-4 shows an example of the resultant window on the “Tera Term” COM console procedure while running the software application on ARM host.



```
COM7 - Tera Term VT
File Edit Setup Control Window Help
initializing axi dma ...
value for dma control register : 11003
setting up SampleGenerator unit...
enabling the interrupt handling system...
performing the first dma transfer... press a key to begin...
Start timer
Measure timer
Execution Time T=11.8sFrame number : 131073
```

Figure 5-4: Tera Term COM console

Focusing on three tables and four graphs (Fig. 5-5, 5-6, 5-7, and Table 5.4, 5.5, and 5.6), we can see that, with an increasing number of packets, the transfer time increases. However, the speed difference between the ACP and HP0 interfaces is very small. The difference is a bit big in the case of GP port, though.

High-Performance Port (HP0):

Table 5.4: Performance Analysis of High-performance port(HP0) For Different Burst Size and PL Clock Frequency

Packet Size(Bytes)	Number of Packets	Total Data Size (Bytes)	PL Clock= 100MHz ; Burst Size = 16bits		PL Clock= 100MHz ; Burst Size = 256bits		PL Clock= 125MHz ; Burst Size = 256bits	
			Time (S)	Speed (MB/s)	Time (S)	Speed (MB/s)	Time (S)	Speed (MB/s)
32	134217728	4294967296	220	19.52	219	19.61	187.2	22.94
128	33554432	4294967296	67	64.10	70	61.36	59.3	72.43
512	8388608	4294967296	30	143.17	32	134.22	28	153.39
2048	2097152	4294967296	20	214.75	18	238.61	15.9	270.12
8192	524288	4294967296	17.2	249.71	12	357.91	10.8	397.68
32768	131072	4294967296	16.3	263.49	11.2	383.48	8.9	482.58

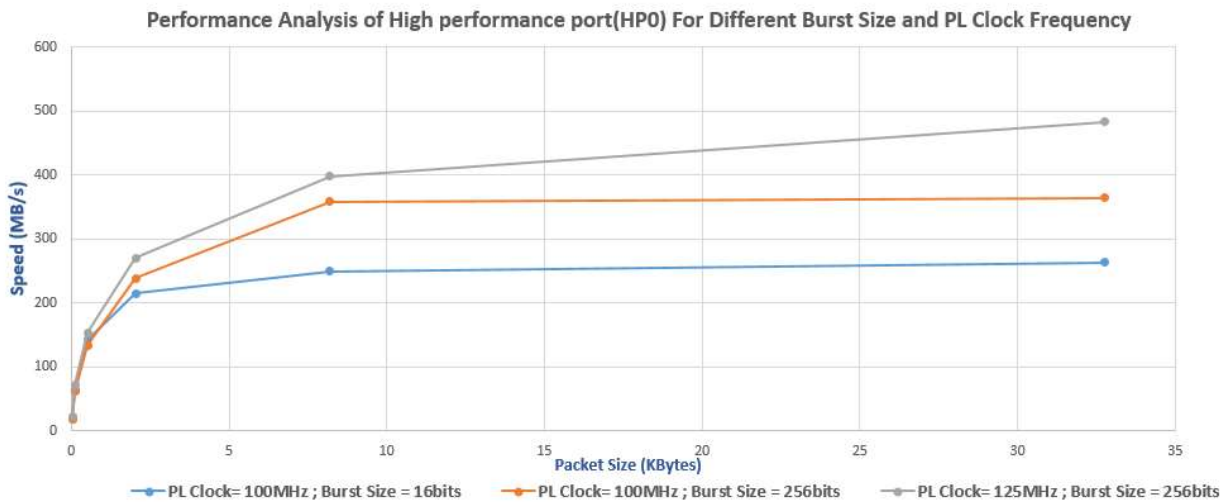


Figure 5-5: Graphical analysis of HP0 interface port For Different Burst Size and PL Clock Frequency

Accelerator Coherency Port (ACP):

Table 5.5: Performance Analysis of Accelerator Coherency Port (ACP) For Different Burst Size and PL Clock Frequency

Packet Size(Bytes)	Number of Packets	Total Data Size (Bytes)	PL Clock= 100MHz ; Burst Size = 16bits		PL Clock= 100MHz ; Burst Size = 256bits		PL Clock= 125MHz ; Burst Size = 256bits	
			Time (S)	Speed (MB/s)	Time (S)	Speed (MB/s)	Time (S)	Speed (MB/s)
32	134217728	4294967296	218	19.70	219	19.61	186	23.09
128	33554432	4294967296	66	65.08	69.8	61.53	58	74.05
512	8388608	4294967296	29	148.10	31.8	135.06	27.5	156.18
2048	2097152	4294967296	20	214.75	19	226.05	15	286.33
8192	524288	4294967296	16.8	255.65	13	330.38	10	429.50
32768	131072	4294967296	16	268.44	11.8	363.98	8	536.87

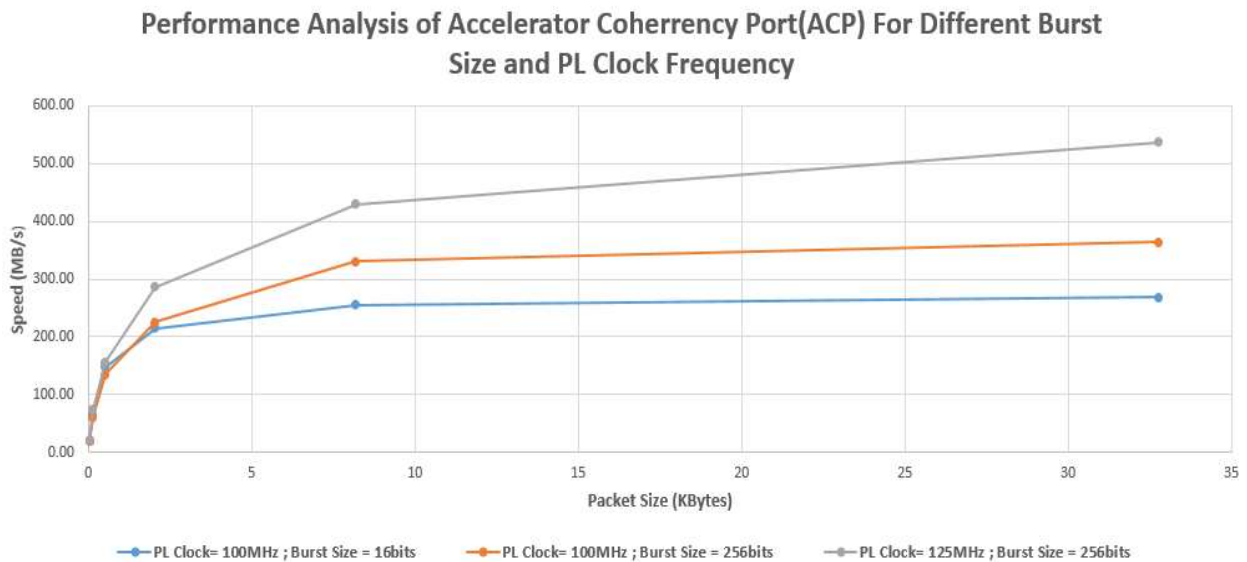


Figure 5-6: Graphical analysis of ACP interface port For Different Burst Size and PL Clock Frequency

General Purpose Port (S_GP0):

Table 5.6: Performance Analysis of General Purpose Port (S_GP0) For Different Burst Size and PL Clock Frequency

Packet Size(Bytes)	Number of Packets	Total Data Size (Bytes)	PL Clock= 100MHz ; Burst Size = 16bits		PL Clock= 100MHz ; Burst Size = 256bits		PL Clock= 125MHz ; Burst Size = 256bits	
			Time (S)	Speed (MB/s)	Time (S)	Speed (MB/s)	Time (S)	Speed (MB/s)
32	134217728	4294967296	462.16	9.29	464.28	9.25	394.32	10.89
128	33554432	4294967296	139.92	30.70	147.976	29.02	122.96	34.93
512	8388608	4294967296	61.48	69.86	67.416	63.71	58.3	73.67
2048	2097152	4294967296	42.4	101.30	40.28	106.63	31.8	135.06
8192	524288	4294967296	35.616	120.59	27.56	155.84	21.2	202.59
32768	131072	4294967296	33.92	126.62	25.016	171.69	16.96	253.24

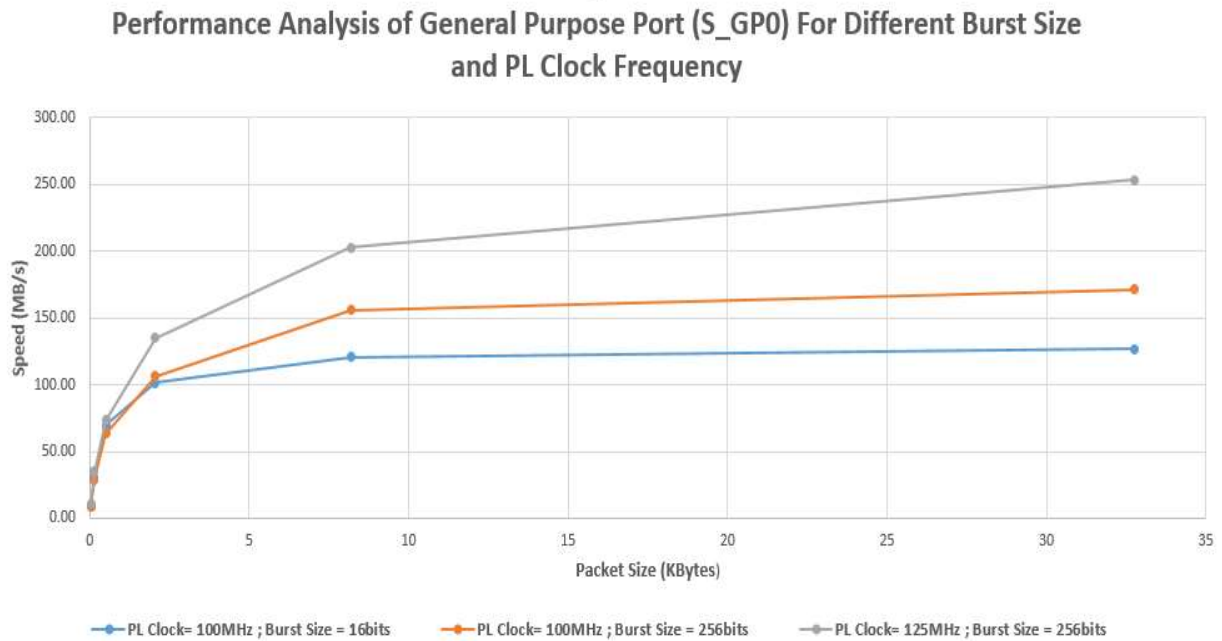


Figure 5-7: Graphical analysis of S_GP0 interface port For Different Burst Size and PL Clock Frequency

5.4.1 Analysis

The performance of three different AXI4-Interfaces has been evaluated in terms of processing time and different traffic conditions. However, for all the three interfaces it can be observed that, whenever the burst size is increased from 16 bits to 256 bits, keeping the sample PL frequency, the speed for transferring the data is increased. Although it has shown some exception with the 128 and 512 bytes of data, overall the transfer speed is improved. However, with the increase of the PL frequency to 125MHz, the speed also increased to about 1.25%. With the increment of the bigger packet sizes the packet numbers decreases hence the speed increases for the reduced traffic.

However, in transferring data from PL to DDR memory this performance might not be the same in all cases. For example, if the DDR memory is engaged in another task, this performance might be changed. Even if the PL frequency is increased, the performance wouldn't change much due to the bottleneck of the memory subsystem.

In terms of the number of packets or the traffic conditions, it can be observed that whenever the number of packets increases the speed or the bandwidth decreases. This is due to the high interrupt rates that are coming from the AXI DMA, which configures the PS after the receipt of each packet.

Hence, with the increment of the packet numbers, the performance is influenced. Among all the three interfaces, ACP port has a better performance compared to HP0 and S_GP0 interface. Compared to the HP0 port, the ACP has almost the same speed, but it showed the best performance among the three. This is due to the connection of the ACP port to the CPU cache, which allows ACP transactions to interact with the CPU cache due to internal connection. This decreases mutual latency for the data that a CPU would be using.

In terms of transferring a bulk amount of data incorporating four HP ports can give a good performance with the increase bus size by four times of one HP port.

6. CONCLUSION AND FUTURE WORK

6.1 Conclusion

This thesis evaluates and analyzes the performance of the three most important AXI interfaces used for PL-PS communication. The results in this thesis are based on the physical implementation of the hardware design using the Vivado design tools on Xilinx Zynq SoC platform. The design is developed with a focus on performance analysis of the AXI interfaces.

Based on the described test concept, execution, and results obtained, it can be concluded that the selection of suitable ports should be carried out based on application requirements. ZYNQ PS-PL interfacing provides a rich variety of interfaces supporting low, moderate, and high data rates. It also provides memory-mapped and stream interface with and without mode addressing. The quality of data rates and bandwidth explains the necessity of this performance analysis. For applications with low bandwidth, the GP port is appropriate. In addition to that, for high-speed applications High Performance (HP) port and Accelerator Coherence Port (ACP) are suitable and work better.

Both interfacing port (HP and ACP) have some advantages and disadvantages. ACP port has the advantage of cache coherency. This ensures the uniformity of shared resource of data which is stroed in different local caches. The ACP links the PL directly with the ARM Cortex-A9 processor's snoop control unit, which allows cache-coherent access to L1 and L2 caches' CPU data. Due to this advantage, it can use less energy for some data processing applications and low iteration time. Due to this reason, ACP can have a high speed compared to HP port. If any other computational function with higher priority is in a parallel process, performance can degrade due to such coherent L2 cache entry. In this case, the HP port is better because it offers the same output in transferring data to DDR memory. HP port is also suitable in transferring a bulk amount of data when all four HP ports are in use.

6.2 Future Work

As a part of future work the performance analysis can also be done on the real-time data transfer, such as by using Ethernet port, the TCP, IP protocol data transfer can also be analyzed. Furthermore, the work that has presented in this thesis work can also be extended by analyzing and reducing the interrupt rate or by using the AXI DMA in scatter-gather mode. Moreover, the Zynq processor can also be used in data acquisition application, and also in RF data transmitter-receiver application and also for Video Processing.



Figure 6-1: Proposed workflow of video processing

This analysis will be a great reference in video processing, as a part of future work for this platform. In order to implement the design for the video processing, the camera module can be connected with the HDMI input or through the serial camera control bus (SCCB) interface. For the further modification, some processing algorithm can be implemented in PL using VIVADO HLS. That cusmoised IP can be exported and integrated with the AXI VDMA. To get the output processed streaming data, the AXI_VDMA need to be connected to HP0 port of the ZYNQ through an AXI interconnect.

APPENDIX A

Source codes of “Sample Data Generator”

```
////////////////////////////////////
// Name: Tilottoma Barua
// Create Date: 10/15/2019 11:47:03 PM
// Design Name: Source code for Sample Data Generator
// Module Name: AXI Stream module_Sample Data Generator
// Project Name: Data Transfer Performance Analysis from PL to PS
// Target Devices: Zedboard –Z7020
////////////////////////////////////

`timescale 1 ps / 1 ps
module sample_generator_v1_0_M_AXIS #
(
    // Users to add parameters here
    // User parameters ends
    // Do not modify the parameters beyond this line
    // Width of S_AXIS address bus. The slave accepts the read and write addresses of
width C_M_AXIS_TDATA_WIDTH.
    parameter integer C_M_AXIS_TDATA_WIDTH = 32,
    // Start count is the numeber of clock cycles the master will wait before
initiating/issuing any transaction.
    parameter integer C_M_START_COUNT = 32
)
(// Users to add ports here
    input wire [31:0] FrameSize,
    input wire En,
    // User ports ends
    // Do not modify the ports beyond this line
    // Global ports
    input wire M_AXIS_ACLK,
    // input wire M_AXIS_ARESETN,
    // Master Stream Ports. TVALID indicates that the master is driving a valid transfer,
A transfer takes place when both TVALID and TREADY are asserted.
    output wire M_AXIS_TVALID,
    // TDATA is the primary payload that is used to provide the data that is passing
across the interface from the master.
    output wire [C_M_AXIS_TDATA_WIDTH-1 : 0] M_AXIS_TDATA,
```

```

        // TSTRB is the byte qualifier that indicates whether the content of the associated
        byte of TDATA is processed as a data byte or a position byte.
        output wire [(C_M_AXIS_TDATA_WIDTH/8)-1 : 0] M_AXIS_TSTRB,
        // TLAST indicates the boundary of a packet.
        output wire M_AXIS_TLAST,
        // TREADY indicates that the slave can accept a transfer in the current cycle.
        input wire M_AXIS_TREADY
    );
    // sample generator - counter
    reg [C_M_AXIS_TDATA_WIDTH-1 : 0] counterR;
    assign M_AXIS_TDATA = counterR;
    assign M_AXIS_TSTRB = {(C_M_AXIS_TDATA_WIDTH/8){1'b1}};
    // counterR circuit
    always @(posedge M_AXIS_ACLK)
        if ( ! M_AXIS_ARESETN ) begin
            counterR <= 0;
        end
        else begin
            if ( M_AXIS_TVALID && M_AXIS_TREADY ) begin
                if ( M_AXIS_TLAST )
                    counterR <= 0;
                else
                    counterR <= counterR + 1;
            end
        end
    // circuit to count number of clock cycles after reset.
    reg sampleGeneratorEnR;
    reg [7:0] afterResetCycleCounterR;
    always @(posedge M_AXIS_ACLK)
        if ( ! M_AXIS_ARESETN ) begin
            sampleGeneratorEnR <= 0;
            afterResetCycleCounterR <= 0;
        end
        else begin
            afterResetCycleCounterR <= afterResetCycleCounterR + 1;

            if ( afterResetCycleCounterR == C_M_START_COUNT )
                sampleGeneratorEnR <= 1;
        end
    // M_AXIS_TVALID circuit
    reg tValidR;
    assign M_AXIS_TVALID = tValidR;
    always @(posedge M_AXIS_ACLK)
        if ( ! M_AXIS_ARESETN ) begin

```

```

        tValidR <= 0;
    end
    else begin
        if ( ! En )
            tValidR <= 0;
        else if ( sampleGeneratorEnR )
            tValidR <= 1;
        end
    end
    // M_AXIS_TLAST circuit
    reg [31:0] packetCounter;
    always @(posedge M_AXIS_ACLK)
        if ( ! M_AXIS_ARESETN ) begin
            packetCounter <= 32'hffffffff;
        end
        else begin
            if ( M_AXIS_TVALID && M_AXIS_TREADY ) begin
                //if ( packetCounter == (FrameSize-1) )
                //    packetCounter <= 32'hffffffff;
                if ( M_AXIS_TLAST )
                    packetCounter <= 32'hffffffff;
                else
                    packetCounter <= packetCounter + 1;
            end
        end
    end
    assign M_AXIS_TLAST = ( packetCounter == (FrameSize-2) ) ? 1 : 0;
endmodule

```

APPENDIX B

Testbench Logic Simulation of “Sample Data Generator”

//

// Name: Tilottoma Barua

// Create Date: 10/15/2019 11:47:03 PM

// Design Name: Source code for Sample Data Generator Test Bench

// Module Name: testbenchsamplegenerator

// Project Name: Data Transfer Performance Analysis from PL to PS

// Target Devices: Zedboard –Z7020

//

module testbenchsamplegenerator();

reg AXI_En;

reg En;

reg [7:0] FrameSize;

wire [31:0] M_AXIS_tdata;

wire M_AXIS_tlast;

reg M_AXIS_tready;

wire [3:0] M_AXIS_tstrb;

wire M_AXIS_tvalid;

reg [31:0] S_AXIS_tdata;

reg S_AXIS_tlast;

wire S_AXIS_tready;

reg [3:0] S_AXIS_tstrb;

reg S_AXIS_tvalid;

reg Clk;

reg ResetN;

initial begin

AXI_En= 0;

FrameSize= 16;

S_AXIS_tdata=0;

S_AXIS_tlast=0;

S_AXIS_tstrb=0;

S_AXIS_tvalid=0;

end

initial begin

Clk= 0;

forever #1 Clk=~Clk;

```

end
initial begin
    ResetN = 0;
    #100 ResetN= 1;
end
initial begin
    En = 1;
    #1000 En= 0;
    #100 En= 1;
end
initial begin
    M_AXIS_tready = 0;
    #200 M_AXIS_tready= 1;
    #2000 M_AXIS_tready= 0;
    #200 M_AXIS_tready= 1;
end
design_1test_wrapper dut
(.AXI_En(AXI_En),
.En(En),
.FrameSize(FrameSize),
.M_AXIS_tdata(M_AXIS_tdata),
.M_AXIS_tlast(M_AXIS_tlast),
.M_AXIS_tready(M_AXIS_tready),
.M_AXIS_tstrb(M_AXIS_tstrb),
.M_AXIS_tvalid(M_AXIS_tvalid),
.S_AXIS_tdata(S_AXIS_tdata),
.S_AXIS_tlast(S_AXIS_tlast),
.S_AXIS_tready(S_AXIS_tready),
.S_AXIS_tstrb(S_AXIS_tstrb),
.S_AXIS_tvalid(S_AXIS_tvalid),
.s_axis_aclk(Clk),
.s_axis_aresetn(ResetN));
endmodule

```

REFERENCES

- [1] Xilinx, “Zynq-7000 All Programmable SoC Overview.” 2013.
- [2] L. H. Crockett, R. A. Elliot, M. A. Enderwitz, and B. Stewart, *The Zynq Book: Embedded Processing with the Arm Cortex-A9 on the Xilinx Zynq-7000 All Programmable Soc*. Strathclyde Academic Media, 2014.
- [3] S. Lafond and J. Lilius. “Interrupt costs in embedded system with short latency hardware accelerators” In Engineering of Computer-Based Systems, 2008. ECBS2008. 15th Annual IEEE International Conference and Workshop on the pages 317–325, 2008
- [4] T. Berg. “Maintaining i/o data coherence in embedded multicore systems”.Micro, IEEE, 29(3):10–19, 2009.
- [5] Sklyarov, Valery, and Iouliia Skliarova. "Exploration of high-performance ports in Zynq-7000 devices with different traffic conditions." In Electrical Engineering-Boumerdes (ICEE-B), 2017 5th International Conference on, pp. 1-4. IEEE, 2017.
- [6] Coseriu, Bogdan, Mihai Negru, and Sergiu Nedevschi. "Contrast restoration of foggy images on the ZYNQ embedded platform." In Intelligent Computer Communication and Processing (ICCP), 2016 IEEE 12th International Conference on, pp. 207-214. IEEE, 2016.
- [7] Zynq, Xilinx. "7000." Zynq-7000 all programmable soc overview, advance product specification-ds190 (v1. 2) available on: <http://www.xilinx.com/support/documentation-/data sheets/-ds190-Zynq-7000-Overview.pdf>,” August (2012).
- [8] Schwiegelshohn, Fynn, and Michael Hübner. "Design of an attention detection system on the zynq-7000 soc." In Re-Configurable Computing and FPGAs (Re-Configure), 2014 International Conference on, pp. 1-6. IEEE, 2014.
- [9] A. Putnam, A. M. Caulfield, E. S. Chung, D. Chiou, K. Constantinides, J. Demme, H. Esmaeilzadeh, J. Fowers, G. P. Gopal, J. Gray, M. Haselman, S. Hauck, S. Heil, A. Hormati, J.-Y. Kim, S. Lanka, J. Larus, E. Peterson, S. Pope, A. Smith, J. Thong, P. Y. Xiao, and D. Burger, “A Reconfigurable Fabric for Accelerating Large-Scale Datacenter Services.” International Symposium on Computer Architecture, 2014.
- [10] Xilinx, Inc., Vivado Design Suite AXI Reference Guide, Xilinx, Inc., 2017, p. 82.
- [11] J. Teich, *Hardware/software codesign: “The past, the present, and predicting the future, Proceedings”*, IEEE 100 (Special Centennial Issue) (2012) 1411-1430.

- [12] C. U. Smith, G. A. Frank, J. Cuadrado, “*An architecture design and assessment system for software/hardware codesign*” , in Design Automation, 1985. 22nd Conference on, IEEE, 1985, pp. 417-424.
- [13] R. Ernst, J. Henkel, T. Benner, “*Hardware-software co synthesis for microcontrollers*”, Readings in hardware/software co-design (2002) 18-29.
- [14] R. K. Gupta, G. De Micheli, “*Hardware-software co synthesis for digital systems, Design & Test of Computers*”, IEEE 10 (3) (1993) 29-41.
- [15] P. H. Chou, R. B. Ortega, G. Borriello, “*The book of hardware/software co-synthesis system*”, in Proceedings of the 8th international symposium on System synthesis, ACM, 1995, pp. 22-27.
- [16] T. Blickle, J. Teich, L. Thiele, “*System-level synthesis using evolutionary algorithms*”, Design Automation for Embedded Systems 3 (1) (1998) 23-58.
- [17] Xilinx: Zynq Architecture (2012), AXI4-Stream FIFO v4.1
- [18] Zynq 7000 SoC: Technical Reference manual
- [19] Xilinx: 7 Series FPGAs and Zynq-7000 SoC XADC Dual 12-Bit 1 MSPS Analog-to-Digital Converter(July 2018)
https://www.xilinx.com/support/documentation/user_guides/ug480_7Series_XADC.pdf
- [20] ARM: AMBA AXI and ACE Protocol Specification (December 2017)
<https://developer.arm.com/docs/di0022/latest/amba-axi-and-ace-protocol-specification-axi3-axi4-axi5-ace-and-ace5>
- [21] Xilinx, Vivado Design Suite User Guide, Implementation, v2013.4 ed., December 2011
- [22] Shalf, J., Quinlan, D., Janssen, C.: “*Rethinking hardware-software codesign for exascale systems*”.Computer 44(11), 22–30 (November 2011)
- [23] C. Rubattu et al., “*Dataflow-Functional High-Level Synthesis for Coarse-Grained Reconfigurable Accelerators*”. IEEE Embedded Systems Letters (2018).doi:10.1109/LES.2018.2882989.
- [24] B. Bougard, B. De Sutter, D. Verkest, L. Van der Perre, and R. Lauwereins, “*A coarse-grained array accelerator for software-defined radio baseband processing*,” IEEE Micro, vol. 28, no. 4, pp. 41–50, Jul./Aug.2008.
- [25] F. Slomka, M. Dörfel, R. Münzenberger, R. Hofmann. “*Hardware/Software Codesign and Rapid-Prototyping of Embedded Systems*”. IEEE Design & Test of Computers, Specialissue: Design Tools for Embedded Systems, Vol. 17, No. 2, April-June 2000.
- [26] T. R. Mück and A. A. Fröhlich, “*Towards Unified Design of hardware and Software Components Using C++*,” IEEE Transactions on Computers, 2013, to appear. [Online]. Available: http://www.lisha.ufsc.br/pub/Muck_TC_2013.pdf

- [27] R. Ernst, B. “*Codesign of embedded systems: Status and trends*”, IEEE Des. Test Comput., vol. 15, no. 2, pp. 45–54, Apr. 1998.
- [28] G. Karsai, F. Massacci, L. J. Osterweil, and I. Schieferdecker, “*Evolving embedded systems*,” IEEE J. Comput., vol. 43, no. 5, pp. 34–40, May2010.
- [29] L. Shang, R. Dick, and N. K. Jha, “*SLOPES: Hardware-software cosynthesis of low-power real-time distributed embedded systems with dynamically reconfigurable FPGAs*”, IEEE Trans. Comput.-AidedDesign Integr. Circuits Syst., vol. 26, no. 3, pp. 508–526, Jul. 2007.
- [30] F. Vahid and T. Givargis, “*Platform Tuning for EmbeddedSystems Design*”,Computer, vol. 34, no. 3, pp. 112-114, Mar. 2001
- [31] R. Tessier, K. Pocek, and A. DeHon, “*Reconfigurable computing architectures*”, Proc. IEEE, vol. 103, no. 3, Mar. 2015
- [32] S. Trimberger, “*Three ages of FPGAs*,”Proc. IEEE, vol. 103, no. 3, pp. 318–331,Mar. 2015.
- [33] Crockett, L.H. and Northcote, D. and Ramsay, C, “*Zynq MPSoc Book – With PNYQ and Machine Learning Applications*”, 2019