CHALMERS
UNIVERSITY OF TECHNOLOGY

UNIVERSITY OF GOTHENBURG

AXI4 Interfaces on a Deep Neural Network Accelerator

Master's thesis in Embedded Electronic System Design

Yuxiang Cao

Department of Computer Science and Engineering
CHALMERS UNIVERSITY OF TECHNOLOGY
UNIVERSITY OF GOTHENBURG
Gothenburg, Sweden 2024

Master's thesis 2024

AXI4 Interfaces on a Deep Neural Network Accelerator

Yuxiang Cao

UNIVERSITY OF
GOTHENBURG

CHALMERS
UNIVERSITY OF TECHNOLOGY

Department of Computer Science and Engineering
Chalmers University of Technology
University of Gothenburg
Gothenburg, Sweden 2024

AXI4 Interfaces on a Deep Neural
Network Accelerator
Yuxiang Cao

© Yuxiang Cao, 2024.

Supervisor: Lars Svensson, Microwave Electronics, Microtechnology and Nanoscience
Company advisor: Jonny Fransson, Imsys AB
Examiner: Per Larsson-Edefors, Microwave Electronics, Microtechnology and Nanoscience

Master's Thesis 2024
Department of Computer Science and Engineering
Chalmers University of Technology and University of Gothenburg
SE-412 96 Gothenburg
Telephone +46 31 772 1000

Typeset in LaTeX
Gothenburg, Sweden 2024

AXI4 Interfaces on a Deep Neural Network Accelerator
Yuxiang Cao
Department of Computer Science and Engineering
Chalmers University of Technology and University of Gothenburg

Abstract

The demand for artificial intelligence in intellectual property and hardware platforms including field-programmable gate arrays and application-specific integrated circuits is rapidly increasing, especially in fields such as computer vision, radar, and image analysis. However, traditional artificial intelligence acceleration methods are facing challenges due to the surge in data volumes, necessitating the development of sustainable computing solutions. Reconfigurable devices, renowned for their high customization, adaptability, and parallelism properties, are emerging as pivotal components in addressing these challenges.

This thesis project investigates the integration of an AMBA AXI4-Lite slave interface into a deep neural network accelerator intellectual property core using VHDL. The primary objective is to optimize communication channels between accelerators and processors, thereby enhancing system configuration accuracy and memory management efficiency. The anticipated outcome encompasses performance enhancements based on the implementation of the AXI4-Lite slave interface. The study extends its scope to include optimize communication pathways and exploring novel approaches to system configurations and memory management within the intellectual property core design.

Keywords: FPGA, VHDL, RTL design, IP core, artificial intelligence.

Acknowledgements

I want to express my sincere gratitude to my supervisors Lars and Jonny, for their guidance, support and knowledge throughout my project. I am also grateful to my examiner Per, for his invaluable advice on my report and presentation.

I want to thank my friend Yuhua for his suggestion which helped me get this thesis topic and Dag from the company that allowed me to work on the project.

A special thanks to my parents and friends for their support and understanding that made me come to Sweden to pursue my master's degree and helped me through the journey.

Thank you all for your contributions to the completion of this thesis.

Yuxiang Cao, Gothenburg, September 2024

Contents

1

Introduction

The integration of deep neural network (DNN) accelerators into existing systems encounters challenges related to command communication and data transfer between the accelerator and the host processor [1]. The choice of control interface between the accelerator and the processor significantly impacts the overall system performance, flexibility, and complexity. These considerations are crucial for accommodating diverse data structures and access patterns inherent in DNN workloads. The integration of DNN accelerators within reconfigurable devices has become pivotal for real-time, resource-efficient inference tasks. Field programmable gate arrays (FPGA) based embedded systems have been researched and proven to be energy-efficient with low latency compared to graphics processors [2] [3]. Additionally, there are examples of CPU-FPGA heterogeneous systems using the Xilinx Inc. Zynq platform [4]. When it comes to a hardware interface, Advanced eXtensible Interface (AXI) protocol is often used to connect components that target high-performance, high-frequency system designs; it includes several features making it suitable for high-speed system interconnects [5]. In this thesis, we will explore the implementation of the command transmission protocol between the accelerator and the processor.

1.1 Related Works

The ubiquity of DNN accelerators in response to the escalating demand for high-performance DNN inference necessitates seamless integration into existing systems. The adoption of the AXI4 protocol facilitates this integration, offering standardized on-chip communication and enhancing the efficiency and flexibility of accelerators. One of the key challenges in developing DNN accelerators is the efficient communication between the accelerator and the processor(s). The AXI4 protocol provides a well-defined and standardized interface that can be used to address this challenge. C. Zhang et al. [6] delve into FPGA-based accelerators for deep convolutional neural networks (CNNs), offering insights into the challenges associated with adapting protocols like AXI4 and AXI4-Lite to FPGA architectures.

Whatmough et al. [7] focus on implementing a powerful fully connected DNN classifier on a 28-nm system-on-chip (SoC) for the Internet of Things (IoT), highlighting the relationship between hardware design and optimal performance. However,

the challenge of not having enough memory bandwidth to match the computation throughput remains a problem, which leads to failure to achieve the best performance. Pagani et al. [8] presented a solution to reserve a given bus bandwidth to each AXI master, but it increases the complexity and resources spent on the FPGA. For Xilinx FPGA, it provides an AXI interconnect called AXI SmartConnect [9]. It defined certain signals to build the system to control the quality-of-service (QoS) of transactions, but failed to solve the conflict among hardware accelerators in the access of the output master port [10].

1.2 Problem Statement

In this project, our primary focus is on developing an AXI4-based slave interface tailored for controlling the DNN accelerator within the context of on-chip communication. The master interface for data transfer has already been established by the company Imsys AB using the AXI4 master, leaving our focus on implementing the AXI4 slave interface for command transfer, depicted in Figure 1.1 by the control interface block. The command interface between the Network on Chip (NoC) and the Noel-V is an AXI4 slave. The Noel-V is a RISC-V processor designed by the company Gaisler. It acts as a master in this case and the NoC serves as a slave, the processor needs to read from or write to specific accelerator addresses via the control interface. The pre-existing AXI4 master lies between the AXI-Data-Acc and the arbiter, transferring the data for double data rate (DDR) and the accelerator. Our approach aims for resource efficiency and streamlined integration.

Our goal is to integrate an AXI4-Lite slave interface into existing system architectures to optimize data I/O operations and enhance communication channels with control processors. The AXI4-Lite interface is a subset of AXI4 without the burst access capability. The design involves providing sufficient bandwidth for high-speed data transfers required in DNN inference while supporting diverse data transfer patterns. The goal is to achieve 100% throughput for command control, while maintaining low energy consumption. We will make a thorough evaluation of the performance and energy cost of the DNN accelerator including simulation and synthesis.

The project objectives include:

- Develop a skid buffer to create a temporary storage entity for the data when a mismatch occurs between sender and receiver.
- Design AXI4-Lite slave writer and reader with the integration of skid buffers to facilitate basic command data transmission.
- Design an interface to achieve data transmission between the accelerator and AXI4-Lite. Test and verify the design under simulation environment with an evaluation focused on power consumption and resource usage.
- Test the design combined with NoC in simulation and synthesis.

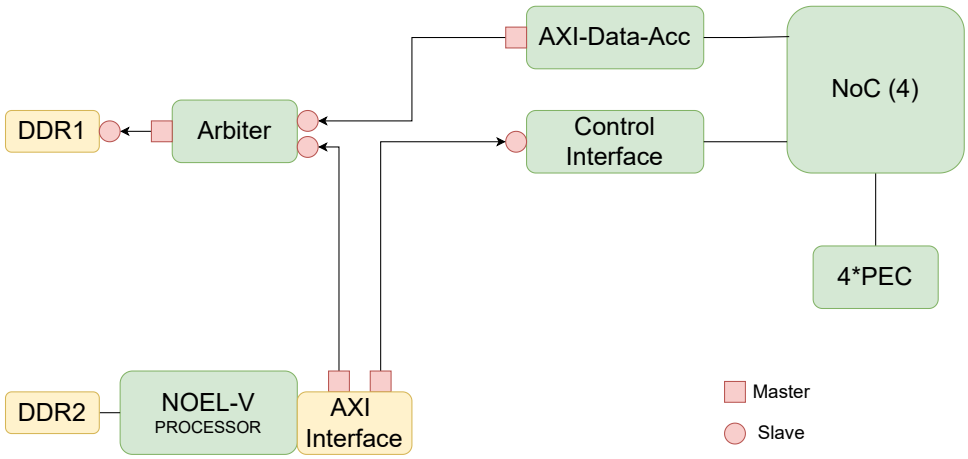

Figure 1.1: The hardware demonstrator for the system[11].

1.3 Delimitations

The resources needed for this project include development software like Vivado, an FPGA for testing and RISC-V processors used in intellectual property (IP) products. The Noel-V processors are developed by another company, to which we do not have access. Our project involves only a part of the DNN accelerator IP, necessitating cooperation with other components, as well as test scripts and testbenches, to evaluate the design.

1.4 Thesis Report Overview

The content of this thesis is divided into five chapters as follows:

1. Introduction: This chapter sets the stage by defining the problem, outlining the research goals, related literature and delimitations of the project.
2. Technical Background: Providing essential context, this chapter covers foundational concepts and methodologies relevant to the research.
3. Method: Outlining the design and test procedures used in this project.
4. Design and Implementation: Detailing the project's design decisions and implementation strategies, this chapter explains the system architecture and technical considerations.
5. Results: Focused on empirical findings, this chapter presents and analyzes results obtained through test and verification, offering insights into the performance of the proposed solution.
6. Conclusion: Concluding the thesis, this chapter reflects on findings, and suggests future research directions.

2

Technical Background

In this chapter, the technical background that was used to develop the project is reviewed.

2.1 Field Programmable Gate Array (FPGA)

FPGAs are configurable integrated circuits that allow the user to alter the behavior of the devices post-production by reprogramming them. This is typically done using a hardware description language such as Verilog or VHDL. Due to this adaptable nature, engineers can model various types of electronic circuits and swiftly modify them, making FPGAs an important tool for electronic system verification and high-performance systems [12].

FPGAs consist of a large number of configurable logic blocks (CLBs), which contain lookup tables (LUTs) and flip-flops [13]. LUTs are programmable components that can represent any desired logic gate, and the flip-flops are sequential elements used for storing data. The CLBs process information concurrently, allowing FPGAs to offer enhanced processing performance compared to conventional CPUs that operate sequentially. The characteristics of FPGAs make them highly beneficial in high-speed tasks, and they are widely utilized in various fields such as digital signal processing and telecommunications.

2.2 Intellectual Property Core (IP core)

IP cores are pre-designed, reusable blocks of semiconductor logic functions, which can be customized and integrated into larger chip designs. Utilizing semiconductor IP cores offers several advantages. It streamlines the chip development process by providing ready-made solutions for common functions, reducing design time and costs [14]. Additionally, IP cores often undergo rigorous testing and optimization, ensuring reliability and performance in the final chip design. IP cores facilitate design flexibility. Designers can modify and optimize these cores to meet specific requirements, tailoring them to the unique needs of their applications. This flexibility enables rapid prototyping and iteration, accelerating time-to-market for new semiconductor products.

IP cores serve as building blocks for innovative chip designs, empowering designers to create complex semiconductor solutions efficiently and effectively. They are a vital component of modern semiconductor design methodologies, driving innovation and advancement in various industries, from consumer electronics to automotive and beyond.

2.3 AMBA AXI4 Protocol

The Arm Advanced Microcontroller Bus Architecture (AMBA) serves as an open-standard on-chip interconnect specification tailored for facilitating the connection and orchestration of functional blocks within System-on-a-Chip (SoC) designs [15]. It defines the communication protocols for multiple functional blocks and components within a bus architecture. AMBA is widely employed in application-specific integrated circuits (ASIC) and SoC designs, playing a crucial role in devices such as smartphones and IoT systems. Its reusability across different requirements, coupled with support from third-party IP products and tools, enhances its efficiency and convenience of use.

The AXI protocol was initially introduced as part of AMBA 3 [16] in 2003 to provide higher-performance interconnects. Subsequently, in 2010, the AMBA 4 [17] AXI4 protocol was introduced.

2.3.1 AXI4

The AMBA AXI4 protocol represents a significant advancement within the fourth generation of the AMBA interface specification, developed by Arm. Within the AMBA framework, three distinct AXI4 protocols are delineated: AXI4, AXI4-Lite, and AXI-Stream. AXI4 facilitates burst mode operations, making it ideal for high-performance memory-mapped data and address interfacing. AXI4-Lite is a subset of AXI4. It presents a simplified interface lacking burst mode capabilities. AXI-Stream has emerged as a swift unidirectional protocol primarily engineered for efficient data transfer from master to slave components within the system.

For AXI4 and AXI4-Lite, five independent channels are defined: read data, read address, write data, write address and write response. The address channels carry the address from which data is read or written. Data is transferred between slave and master through these channels. The write data channel allows data to be transferred from master to slave, while the write response channel indicates the success or failure of the write operation. The read data channel allows the master to read from the slave. Figure 2.1 and Figure 2.2 illustrate read and write transactions.

The most crucial feature of AXI4 is the handshake process. All channels of AXI4 utilize **VALID/READY** signals to control the data transmission. The sender gen-

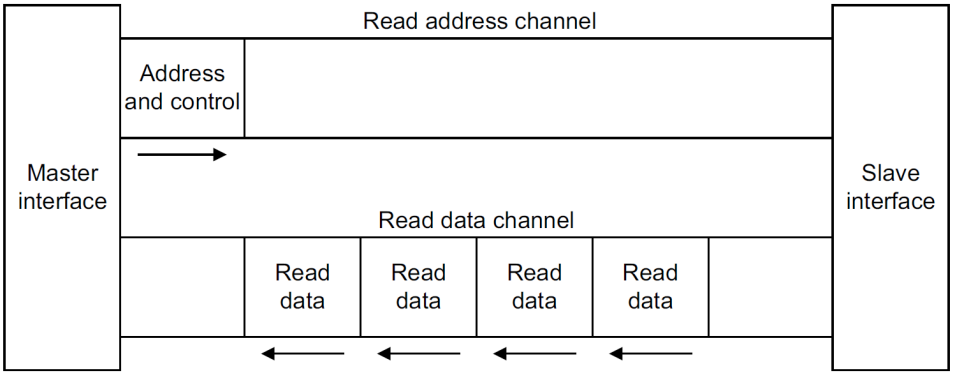

Figure 2.1: The channel architecture of reads [18].

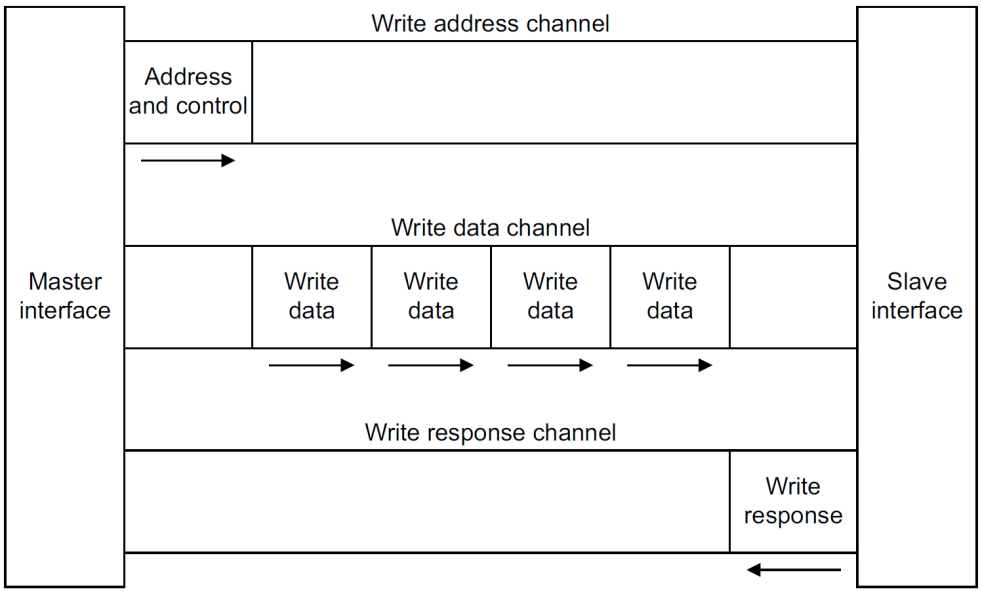

Figure 2.2: The channel architecture of writes [18].

erates the **VALID** signal to indicate the availability of data or commands, while the receiver generates the **READY** signal to indicate its readiness to receive data or commands. A transfer is successful only when both **VALID** and **READY** signals are high. Once **VALID** is asserted, it must remain high until the handshake is a success. The sender must not wait for a high **READY** to assert **VALID**.

Dependencies exist between the channel handshake signals, as depicted in Figure 2.3, 2.4, and 2.5. Single-headed arrows indicate signals that can be asserted depending on previous signals, while double-headed arrows indicate signals that must be asserted depending on previous signals. Figure 2.3 illustrates the dependencies for read transactions, where the slave must wait for **ARVALID** and **ARREADY** to be asserted before asserting **RVALID**. Figure 2.4 shows the write transaction handshake signal dependencies and Figure 2.5 shows all the AXI4 required slave write

response handshake dependencies.

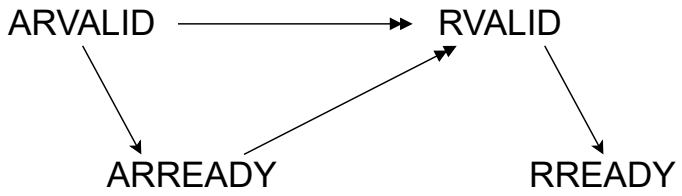

Figure 2.3: Read transaction handshake dependencies [18].

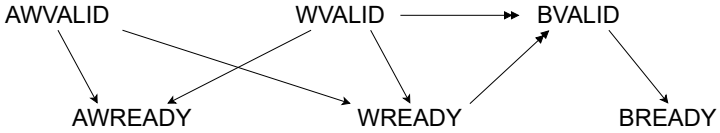

Figure 2.4: Write transaction handshake dependencies [18].

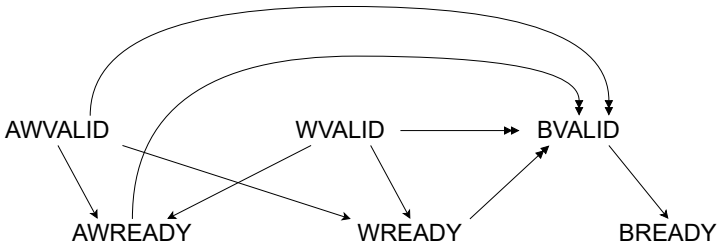

Figure 2.5: Slave write response handshake dependencies [18].

The AXI4 protocol is burst-based initiated by the master and responded to by the slave to calculate the addresses. Three types of burst are defined: fixed burst, incrementing burst and wrapping burst. In a fixed burst, every transfer has the same address, suitable for repeated accesses to the same location, such as using a First-In-First-Out (FIFO). In an incrementing burst, the address for each transfer is an increment of the previous one, with the size of the increment depending on the transfer size. The wrapping burst is similar to the incrementing burst but wraps around to a lower address when a wrap boundary is met.

AXI4 slaves can be classified as memory slaves or peripheral slaves. Memory slaves need to be able to handle all types of transactions, whereas peripheral slaves are only required to work with certain access methods, as long as the erroneous access does not cause any system deadlock.

2.3.2 AXI4-Lite

The AXI4-Lite interface is a subset of AXI4 intended for controlling command registers in components. The AXI4-Lite requires fewer signals and simplifies the design

and verification process. The main difference between AXI4 full and AXI4-Lite lies in their support for burst mode. In AXI4-Lite, all transactions have a burst length of 1, and it supports shorter data widths of either 32 or 64 bits. A bridge is needed when connecting AXI4 full and AXI4-Lite interfaces.

2.4 Double Data Rate (DDR)

Double Data Rate (DDR) is a type of synchronous dynamic random-access memory (SDRAM) that enables high-speed data transfer rates by allowing data to be transferred on both the rising and falling edges of the clock signal [19]. DDR memory modules have become ubiquitous in modern computing devices, including computers, servers, and consumer electronics.

The key innovation of DDR memory lies in its ability to transmit data twice as fast as traditional SDRAM. By utilizing both the rising and falling edges of the clock signal, DDR memory effectively doubles the data transfer rate compared to single data rate (SDR) memory technologies. DDR memory plays a crucial role in modern computing systems, where high performance and reliability are paramount. Its widespread adoption in a variety of applications, from personal computing to data centers, underscores its importance in enabling the rapid and efficient processing of data in today's digital age.

2.5 Network-on-Chip (NoC)

A Network-on-Chip (NoC) is a communication architecture that interconnects various components or processing elements within a single integrated circuit or SoC [20]. It provides a scalable and efficient way to transfer data between different modules or cores, facilitating high-speed and low-latency communication in complex integrated systems.

One of the key advantages of NoC is its flexibility and configurability. Designers can customize the NoC topology and routing algorithms to meet the specific requirements of their application, optimizing for factors such as power consumption, performance, and fault tolerance. The NoCs can support different types of communication protocols and varying bandwidth requirements for different modules or cores. This enables efficient resource sharing and utilization within the system.

3

Method

The development flow of this project can be divided into five general steps: interface determination, RTL design, functional verification, Vivado synthesis and analysis. From the general steps we can derive twelve steps as shown in Figure 3.1 shows the overall workflow of the project.

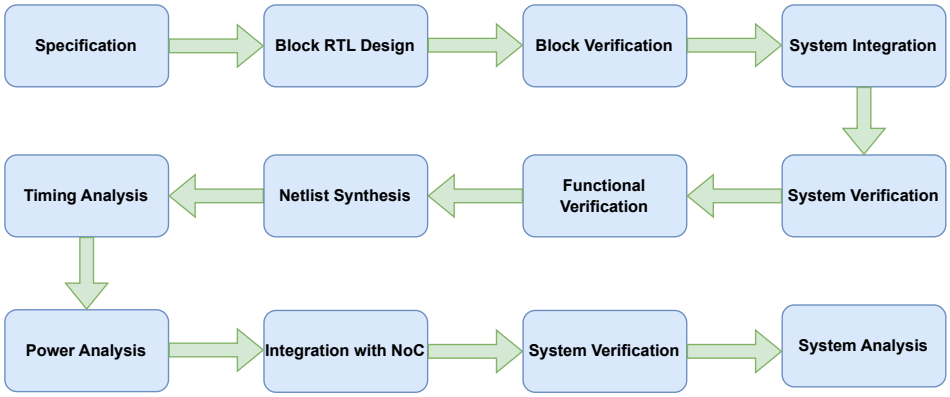

Figure 3.1: The development flow of the project.

3.1 Interface Determination

Several protocols can be selected for this slave interface: AXI4, AMBA Advanced High-performance Bus (AHB) [21] and Advanced Peripheral Bus (APB) [21]. By studying the specifications of above protocols, and considering the system needs and requirements of other components, an interface will be chosen that suits the system best.

3.2 RTL Design

The RTL design is separated into three parts: reader and writer channels for the AXI4-Lite and an interface that collects and sends data from the above two modules and communicates with an external NoC component. The three blocks were designed with customized features for the needs of the system using VHDL. The

data width and address width are both 32 bits. The interface to NoC collects data and integrates it into a package and sends it to NoC whenever the handshake is fulfilled. Similarly, the data required by the processor from the NoC will be sent through reader channel. To facilitate the functional test for different components, testbenches are also developed independently. A system wrapper is generated to pack components.

3.3 Functional Verification

Functional verification is done with Modelsim and Vivado [22]. The version of Modelsim is Intel FPGA starter edition 2020.3 and for Vivado Design Suite is 2022.2. The simulation is mainly done in Modelsim. The source files and testbench files are added to the project together with test scripts. Three components are tested separately first to check if they can process input/output and stall conditions correctly. After integration, we ensure read and write transactions can happen at same time due to the full-duplex property of AXI4. The test with NoC is partly done in Modelsim and Vivado, testbench for NoC is provided but has some issues, due to bankruptcy of the company not enough support can be given.

3.4 Vivado Synthesis and Analysis

All the VHDL codes are synthesized to RTL netlists in Vivado using ZYNQ 7020 FPGA design kit [23]. Utilization and power reports are generated to enable an analysis of the timing and resource consumption. Bitstream cannot be generated because the I/O ports on the FPGA are too few for our design thus simulation provides the main results for the project. A test environment is also built to connect the AXI4-Lite slave withthe AXI4 master.

4

Design and Implementation

In this chapter we will discuss the design and implementation considerations.

4.1 Interface Determination

Three protocols lie as the potential choice for the slave interface of the system: AXI4, AHB and APB. They are all part of the AMBA bus which are on-chip communication protocols that are widely used in SoC designs.

The comparison between AXI4, AHB and APB is shown in Table 4.1. Bandwidth for this slave interface is 32-bit so all three meet the requirement. In the aspect of performance, AXI4 has the highest performance and can operate under higher clock frequency. One thing to be noticed is that the AXI4 with skid buffers has lower latency and higher throughput [24]. APB is specially optimized for connecting low-power, low-bandwidth peripherals such as universal asynchronous receiver-transmitter (UART) [25] to the rest of the system. Since power consumption is not the crucial determinant for the design, all three can be further considered. AXI4 has the most channels: one read address channel, one write address channel, one read data channel, one write data channel and one write response channel. Both APB and AHB have two independent channels for data transfer, which makes AXI4 more complex to implement.

The support for multiple outstanding transactions and out-of-order transactions makes AXI4 stand out. Multiple outstanding transactions refer to the ability of a bus protocol to handle several transactions simultaneously without waiting for one to complete before starting another. This is beneficial in pipelined architectures where multiple stages of transactions can be processed concurrently, reducing idle time and increasing efficiency. Out-of-order transactions mean the system can complete transactions in a different order than they were initiated. This allows for more flexibility in handling transactions, especially in optimizing the use of resources. In AXI4, out-of-order transaction support allows a slave to respond to requests in an order different from that in which the master issued them. In our system, the need to send or receive multiple data at the same time exists, which makes AXI4 the better option.

Table 4.1: Comparison between AXI4, AHB and APB.

	AXI4	AHB	APB
Bandwidth	Up to 256-bit data transfers.	Up to 64-bit data transfers.	Up to 32-bit data transfers.
Performance	High-performance applications and can support multiple outstanding transactions, allows efficient pipelining of data transfers.	High-performance bus, but has lower performance than AXI.	Low-bandwidth, low-power applications.
Power consumption	Power-efficient.	Power-efficient.	Optimized for low-power consumption.
Complexity	Most complex, with more signals and features.	Simpler than AXI but more complex than APB.	The simplist.
Applications	High-Performance Systems, memory interfaces, interconnects in SoCs or high-throughput data transfer.	For mid-range applications like microcontrollers.	For low power applications like sensors and audio codecs.
Number of channels	Five independent channels	Two independent channels	Two independent channels
Support for QoS	Yes.	No.	No.
Exclusive transfers	Supported.	Not supported.	Not supported.
Out-of-order transaction	Supported.	Not supported.	Not supported.

4.2 General Considerations

The AXI4 protocol serves as the backbone of our communication infrastructure, offering the requisite flexibility to accommodate intricate interconnect architectures. However, the adoption of the AXI4 protocol introduces a spectrum of challenges, particularly in the dynamic landscape of virtualized environments where processors execute applications with varying degrees of isolation.

Furthermore, a critical aspect of our project involves consideration of the interface between the accelerator's current and future data transport features. Defining the core requirements underlying this FPGA-specific interplay assumes great importance, ensuring not only optimal performance but also judicious resource utilization. Through a comprehensive analysis of these elements, we aim to delineate a roadmap towards the seamless integration of the slave interface within the FPGA ecosystem, thereby advancing the efficiency and efficacy of on-chip communication paradigms.

Considering the similarity between AXI4 and AXI4-Lite, and the advantage of AXI4-Lite in command control, implementing the AXI4-Lite slave initially and keeping the possibility of optimizing it to a full AXI slave is a preferred approach. For the command transaction, the burst property of full AXI4 is not a necessity. The compatibility of AXI4-Lite and full-duplex mode of communication makes it a good fit for the system. Designing a customized AXI4-Lite slave that fully supports the system consumes less time and effort, and significantly decreases the difficulty of testing and verification.

The existing AXI4 master is utilized for data transfer, and a bridge is necessary for an AXI4-Lite slave to communicate with the master. In this slave interface, only the transfer of commands to specific addresses is required; therefore, the burst feature of AXI4 is unnecessary.

4.3 Design of the Skid Buffer

Skid buffers enable seamless data transfer by temporarily storing data until alignment is achieved, thereby ensuring coherence and integrity in the transmission process. Without skid buffers, we can only have a limited performance of throughput without violating the AXI4 specification. The signals that stalled in the skid buffer will be registered, so no combinational logic is directly applied between inputs and outputs.

To use skid buffers in the AXI4 slave, VALID and READY are included for both inputs and outputs. When they are not asserted at the same time, the data under transmission needs to be stalled in the next clock cycle. No data will be lost during the stall.

When there is no stall, the skid buffer functions as a pass-through. However, if the output is stalled, the incoming data will be copied into the internal register sequentially, as depicted in Figure 4.1. In the subsequent clock cycle, the data stored in the register can be output as shown in Figure 4.2.

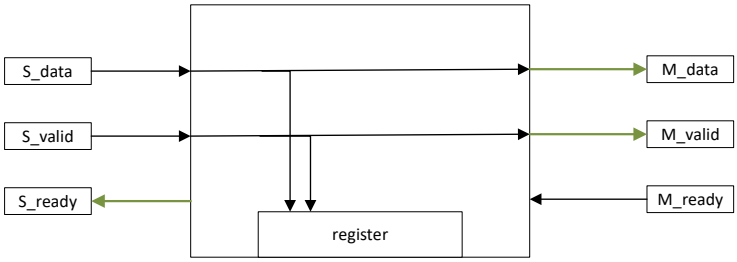

Figure 4.1: Copying data into register.

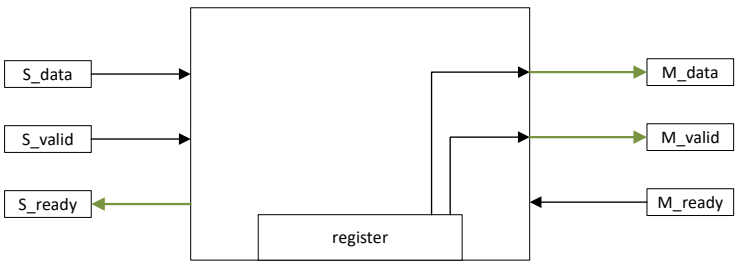

Figure 4.2: Propagation of signal upstream.

4.4 Design of AXI4-Lite Slave

The overall design flow chart of AXI4-Lite is shown in Figure 4.3. The development of it involves three main components: the reader, the writer, and the interface to NoC.

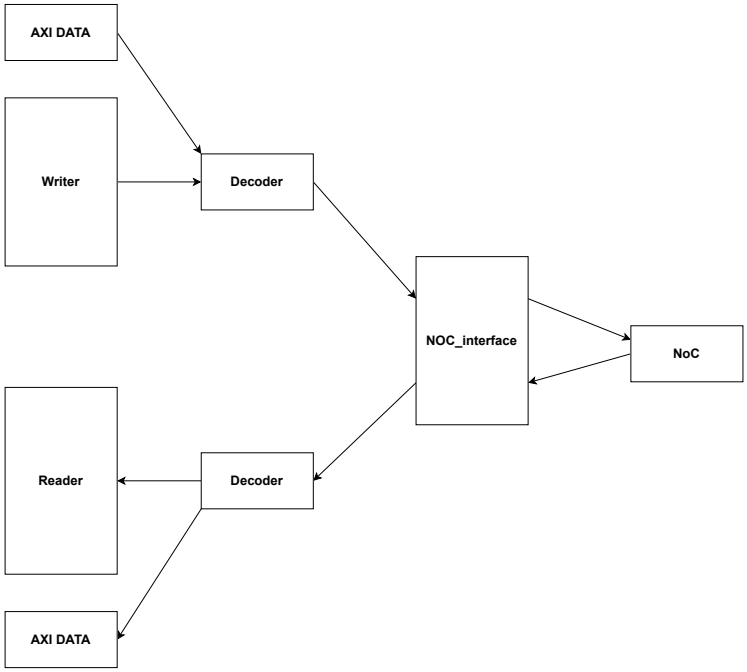

Figure 4.3: The AXI4-Lite design.

4.4.1 Design of the AXI4-Lite Writer

The AXI4-Lite writer is designed to allow the processors to write commands to certain addresses to the accelerator. Figure 4.4 shows the block diagram of the writer component.

Following the AXI4-Lite write data, write address and write response channel specification, three channels are integrated into the writer. The address width and data width are both 32-bit. The processor sends addresses through the bus, which are decoded and distributed to the writer. The addresses to which data is written include four destinations: 0x60, 0x64, 0x68, and 0x6c. Any other address will lead to no response from the interface. Upon detecting the address to which data is written, the writer generates an enable signal accordingly. The sequence of the incoming address and data is not fixed except the address 0x6c always comes last. As long as the address 0x6c is decoded and written to register 3, a last_word signal is then asserted indicating a successful write. Acc_cmd_valid is then pulled to high and processed in the interface to NoC for further action. The four groups of 32-bit data will pass to the interface to form a 128-bit data package, which is subsequently

transmitted to the NoC.

Two skid buffers are used in the writer, one for the write address channel and one for the write data channel with two sets of handshaking rules. They can thus deal with stalls successfully and have higher efficiency. With skid buffers the two channels can keep accepting data unless the output of the interface is stopped. The complexity of the writer and reader is reduced by introducing the skid buffer, as the most important handshaking and stall situation is handled by it.

The S_AXI_BRESP of the response channel in the writer is set to a mode that always sends 00 back to the processor, indicating the slave is always accepting data. S_AXI_BVALID on the other hand, will send back the status of each write.

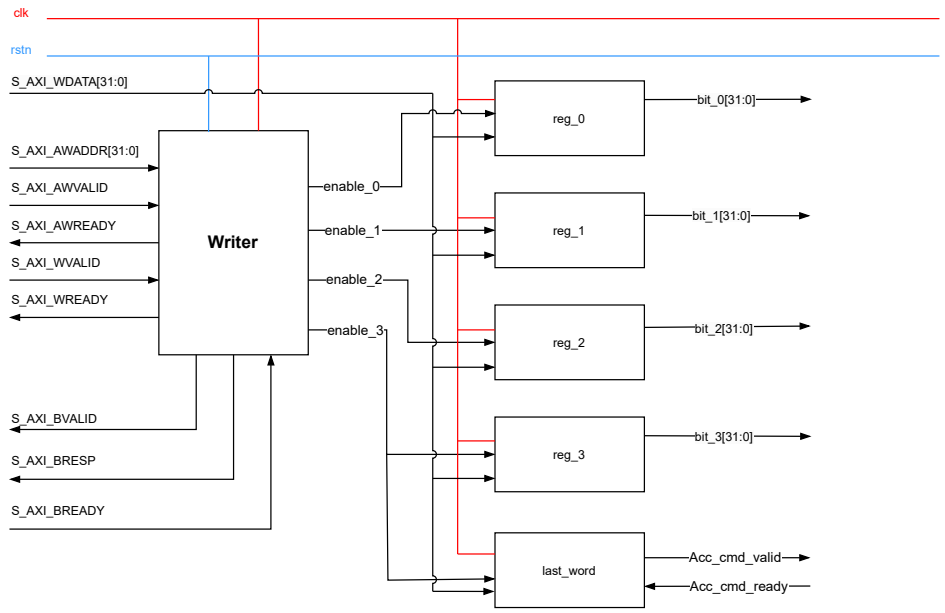

Figure 4.4: The AXI4-Lite writer.

4.4.2 Design of the AXI4-Lite Reader

Similarly to the writer, the AXI4-Lite reader component allows the processor to read from the accelerator. Figure 4.5 shows the block diagram of the reader component.

The reader component has two channels: read address and read data. The address width and data width are also 32-bit. However, the read data transmitted from the NoC is 8-bit width, therefore 24 digits of 0s are added at the most significant bit (MSB) end of data to form a 32-bit data that the processor needs. The reader can read from one additional address, 0x70, in addition to the four addresses mentioned above. When the master sends read address information, it is decoded as in the writer. Upon receiving address information from the master, the reader generates

an appropriate enable signal and reads from the corresponding address. If any addresses other than the five specified above are detected, the reader returns nothing.

The reader has one channel less than the writer, and only one skid buffer is needed for the read address channel. The read data channel does not need it because only one set of data will be read simultaneously, consequently, no stall would happen for the read data channel.

The reader has fewer response signals compared to the writer. For the S_AXI_RRESP it is set to 00 by default indicating that the master is always open for the read option.

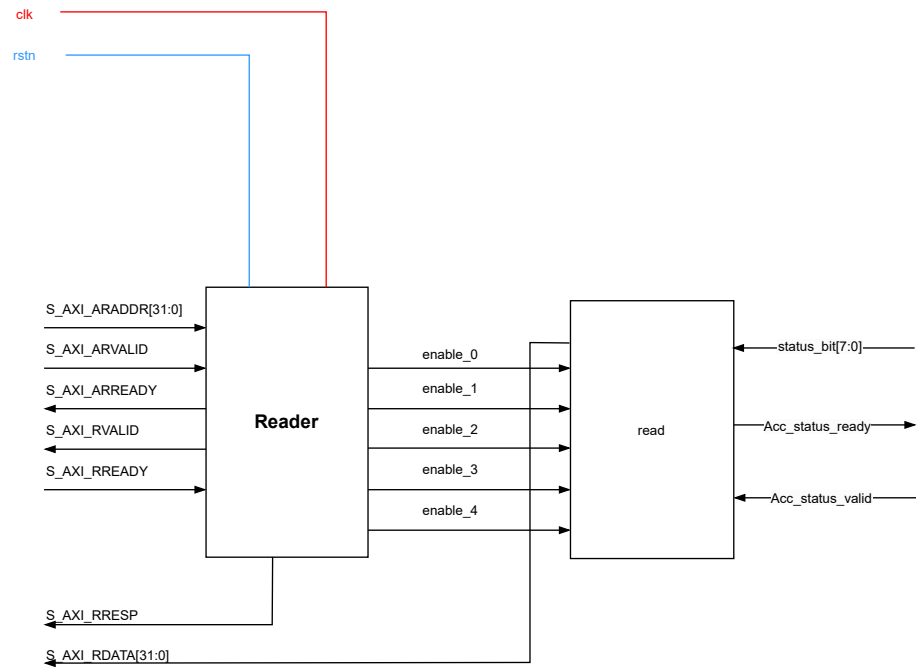

Figure 4.5: The AXI4-Lite reader

4.4.3 Design of the Interface to NoC

The interface to the NoC is responsible for controlling communication with the NoC by transferring both data and status information. Figure 4.6 shows the block diagram of the interface.

The interface aggregates the four groups of separate 32-bit data from the writer into a 128-bit package, ensuring proper transmission to the accelerator. This 128-bit data follows the sequence [31:0], [63:32], [95:64], and [127:96], from addresses 0x60, 0x64, 0x68, and 0x6c respectively. After the last 32 bits are detected, control signals ready and valid from the previous register will control the handshaking and indicate

that new data is ready to be sent. The valid signal remains high until the ready signal is asserted high, ensuring that data modifications in the NoC interface occur only when the valid signal is low. Once the 128-bit data is ready, the valid signal GPP_CMD_Flag will be raised to high and wait until ready signal NOC_CMD_ACK to be asserted from the NoC to cancel the valid.

For the reader side 8-bit status information from the accelerator are commands given to the processor. The valid signal NOC_CMD_Flag follows data and wait for the handshaking with ready signal GPP_CMD_ACK.

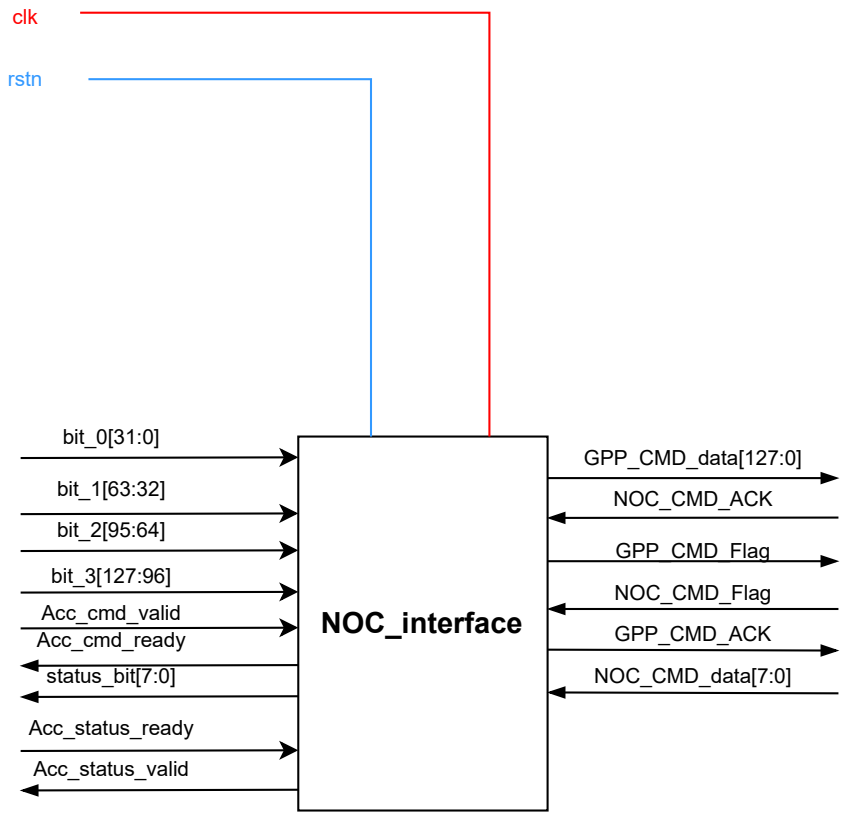

Figure 4.6: The AXI4-Lite interface.

4.5 Testing

Each individual component was tested separately with its own testbench. After that they were integrated into a wrapper which was also tested with dedicated testbenches. Testing the wrapper ensured that the components could successfully interface with each other. Due to the need to integrate our design with designs from other developers, a system-wide testbench is also made.

4.5.1 Testing Individual Components

A well-defined testing methodology was fundamental to ensure that all the components satisfied the same quality criteria. A component was finished when it had successfully and correctly completed a post-synthesis behavioral simulation.

For the skid buffer, the focus was on its ability to process stall and handle handshaking correctly. Test vectors were generated to cover different cases. Such as when a slave asserts ready but the master has not been ready or vice versa. Another possibility is that the master keeps sending data but the slave can not receive that much at the same time, the skid buffer needs to make sure no data is lost during the stall. The test of the skid buffer is important as it is reused as a component in AXI4 writer and reader.

The test for the AXI4 reader and writer was similar. The testbenches imitate situations in which data and addresses were sent in different orders, to see if the data is received in the correct way and at the right time. It is also tested if the addresses were not the intended then no data should be transmitted.

For the interface to NoC, the focus was on whether it could accumulate data from the writer in the correct sequence. Command signals with NoC were also generated by the testbench.

4.5.2 Testing the Wrapper

The testing of the system wrapper is a crucial part of the project to ensure the reliability and functionality of the design. The testing of the system wrapper, including the NoC interface, AXI4 writer, and AXI4 reader, was comprehensive and covered all critical aspects of functionality, performance, and error handling.

The system wrapper was integrated with the interface to the NoC, AXI4 writer and reader. Initial integration tests were conducted to verify signal connectivity and proper data flow. Testbench scripts were used to simulate data patterns typical of the target application. Multiple write and read transactions were initiated to various memory addresses to ensure correct data storage. Verification was achieved by reading back the data and comparing it against expected values. To verify the ability of channels to work independently, multiple reads and writes were issued at different sequences to check if data can be accessed at the right pattern. Synthesis and post-synthesis functional simulation were run with different clock frequencies to check utilization and timing reports.

The initial plan was to test the system on the FPGA board that the company would provide. However, due to their bankruptcy, no FPGA or Vivado licences were available to do the test. A backup plan was to test with Integrated Logic Analyzer (ILA)

and (Virtue IO) VIO IP of Vivado because the IO ports of an available FPGA were not enough to cover all signals. After implementing this method, the IO ports of the FPGA board were still too few for our design.

4.5.3 Testing with NoC Accelerator

The goal of this project was to integrate our interface into the DNN accelerator, so testing this design with other components of the system is important. The NoC component was provided by the company with RTL designs and a general testbench file that was used to test the NoC. Based on the testbench file our design was included as sub-components, modifications of the testbench were also made to make sure our design fit the pipelined data flow. Tested vectors were read by the NoC testbench and outputs were checked with those vectors by making sure no assertions were made by the simulation.

4.5.4 Test Environment in Vivado

To facilitate the testing of the integration of the AXI4-Lite interface with the system, a test environment is built in Vivado with AMD Virtex UltraScale+ FPGA VCU118 Evaluation Kit [26]. With the IP integrator in Vivado, the test environment allows us to connect the AXI4-Lite with the master processor and another AXI4 master interface. Figure 4.7 shows the block design. It contains one AXI crossbar, which is used to connect the AXI memory-mapped masters to another similar memory-mapped slave [27]. Following the crossbar are two AXI clock converters, which are used to connect the master and slave in different clock domains, as the working clock frequency of the AXI4 slave is different from the processor. An AXI data width converter is connected to one of the clock converters, as the data and address width of the AXI4-Lite slave and AXI4 master are different. Finally, an AXI protocol converter which connects one AXI4 master to one AXI4 slave of a different AXI memory-mapped protocol is built.

One thing to be noticed is that this test environment is not used for the final test. At that time the licence for the board was unavailable so the test with the processor component was not conducted.

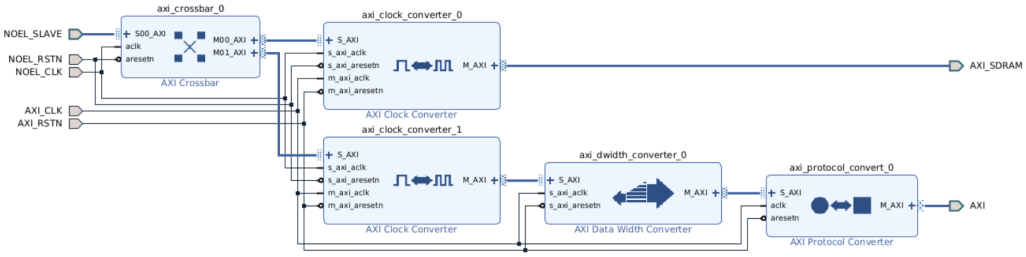

Figure 4.7: The AXI4-Lite test environment in Vivado.

5

Results

In the following chapter results from the evaluation of our design are presented. The evaluation was primarily performed using simulation and synthesis. Analysis of the results is also included in this chapter.

5.1 Simulation Results

Functional verification was conducted in Modelsim to check if the AXI-Lite interface could read and write data from correct addresses respectively with the right sequences. Output results from the simulation were compared with inputs under different circumstances. Signal write and signal read situations were basic requirements of the design, and both tests passed. Write and read to invalid addresses were also tested, resulting in the interface transmitting nothing as we expected. Throughout the simulation, the focus was on the proper use of valid/ready handshaking, correct timing of signal assertions, and appropriate handling of back-to-back transactions. The results showed that the design met the criteria mentioned above.

The design is a customized slave interface so besides the AXI4-Lite part, the functional verification of interaction with external components was also performed. The 128-bit output data to NoC was verified to have the correct sequence from addresses 0x60, 0x64, 0x68 and 0x6c respectively. The handshaking signals with NoC were also checked, the output valid can be deasserted when input ready was detected. The 8-bit read data from NoC was received and formed into 32-bit output data.

5.2 Synthesis Results

In this section timing analysis, utilization report and power analysis are discussed. The design was intended to be tested with a high-performance FPGA board at a clock frequency of 300 MHz. However, due to the company's bankruptcy, no resources from the company were available. The testing FPGA was a development board with limited resources and the maximum clock frequency was 50 MHz. Instead of running at a fixed frequency we decided to test under different clock cycle constraints.

5.2.1 Timing Analysis

Here, we present and analyze the timing summary data obtained from Vivado synthesis across a sequence of chosen clock cycle constraints. The key parameters analyzed include Worst Negative Slack (WNS), Total Negative Slack (TNS), and the number of failing endpoints across setup, hold, and pulse width constraints.

Setup timing is critical for ensuring that the data is correctly captured by the clock signal. The WNS for setup timing indicates the most significant timing violation. Figure 5.1 summarizes the WNS and TNS for the setup timing across the given clock cycle constraints. From the graph, it can be observed that the WNS improves as the clock cycle constraint increases, becoming non-negative from 2.69 ns onwards. TNS follows a similar trend, becoming zero at 2.69 ns, indicating no timing violations beyond this point. The number of failing endpoints (not shown on the plot) decreases significantly, reaching zero from 4 at 2.69 ns. The setup timing analysis revealed that the design meets the setup timing requirements for clock cycle constraints of 2.69 ns and above. For constraints below this threshold, significant timing violations were observed.

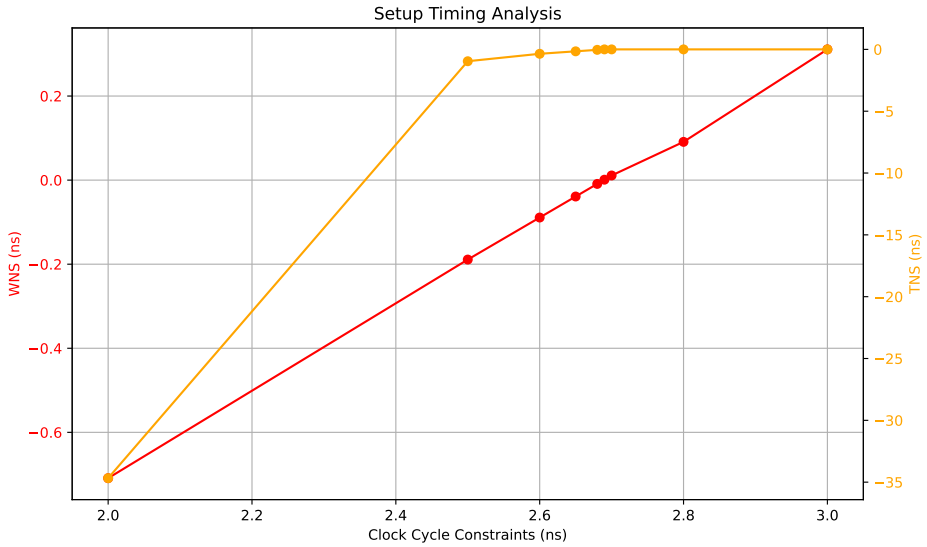

Figure 5.1: Plot of setup timing analysis.

Hold timing ensures that data is held long enough for proper capture by the clock signal. The hold timing shows a consistent WNS of 0.125 ns for all constraints up to 20 ns, demonstrating no hold violations across these constraints. Both the total negative slack and the number of failing endpoints remain at zero for all constraints, confirming that the design meets the hold time requirements throughout the range of clock cycle constraints tested. This consistency across all constraints shows that the design is inherently robust against hold timing violations, ensuring stable data

capture for hold requirements.

Pulse width violations occur when the width of the clock pulse is insufficient for reliable operation. Figure 5.2 shows the worst negative slack, total negative slack for pulse width. The pulse width analysis shows that the WNS improves steadily with relaxed constraints. Starting from 0.408 ns at 2 ns, it increases to 0.750 ns at 2.5 ns, 0.800 ns at 2.6 ns, and continues to improve up to 9.5 ns at 20 ns. This indicates that the pulse width timing becomes more favourable with relaxed constraints. TNS and the number of failing endpoints were zero, confirming the absence of pulse width timing violations. This suggests that the clock pulses are adequately wide for the circuit to function correctly.

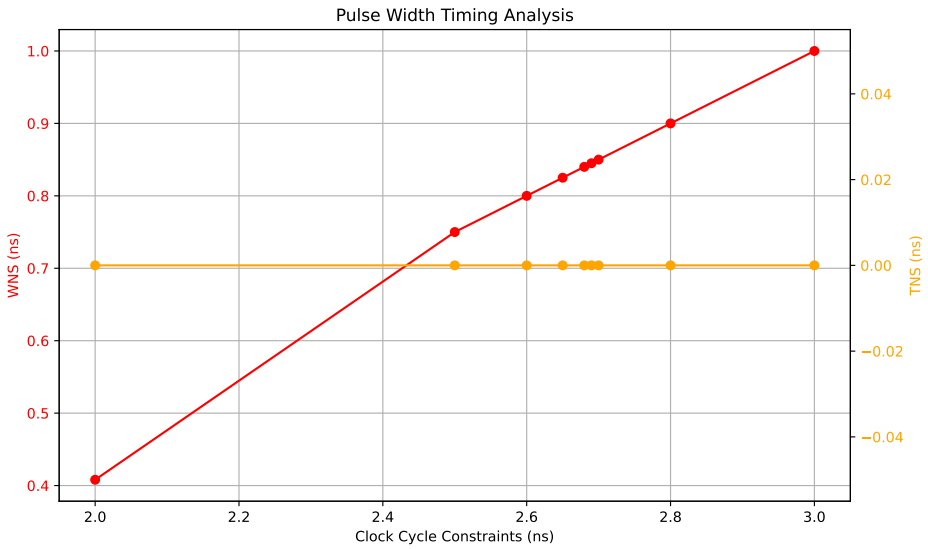

Figure 5.2: Plot of pulse width timing analysis.

5.2.2 Utilization Report

The utilization report shows that for clock cycle constraints from 2 ns to 3 ns, the Slice LUTs utilization remains constant at 159. At 4 ns, the utilization drops to 142, indicating a slight reduction in resource usage. The utilization returns to 159 for a 5 ns constraint and from 10 ns onwards, the Slice LUTs utilization decreases significantly to 107 and remains constant at this value. This variation indicates that the design becomes more resource-efficient in terms of Slice LUTs as the clock cycle constraint is relaxed, especially beyond 10 ns. The utilization of slice registers, bonded Input/Output Blocks (IOBs) and Global Clock Buffer Control (BUFGCTRL) remains constant across all constraints at numbers 616, 284 and 1 respectively, suggesting no dependency on clock cycle constraints. The bonded IOB exceeded the number that our FPGA board has.

5.2.3 Power Analysis

Power consumption is of vital importance in circuit design. As seen in Figure 5.3, where the power consumption drops from 0.351 W at 2 ns to 0.129 W at 20 ns. As the power consumption of other components in the DNN system is unknown, it is hard to decide if our design is power efficient compared to other components in the system.

The dynamic power shows a similar decreasing trend. Starting at 0.243 W at 2 ns, it reduces to 0.024 W at 20 ns. The primary reason for the reduction in total on-chip power is the decrease in switching activity. As clock cycle constraints increase, the design operates at a slower clock speed, leading to fewer signal transitions per unit time. Relaxed constraints often allow for simplifications in the design, reducing the number of active elements and thus the overall power. The decrease in power consumption with relaxed constraints indicates a reduction in switching activity and overall power requirements.

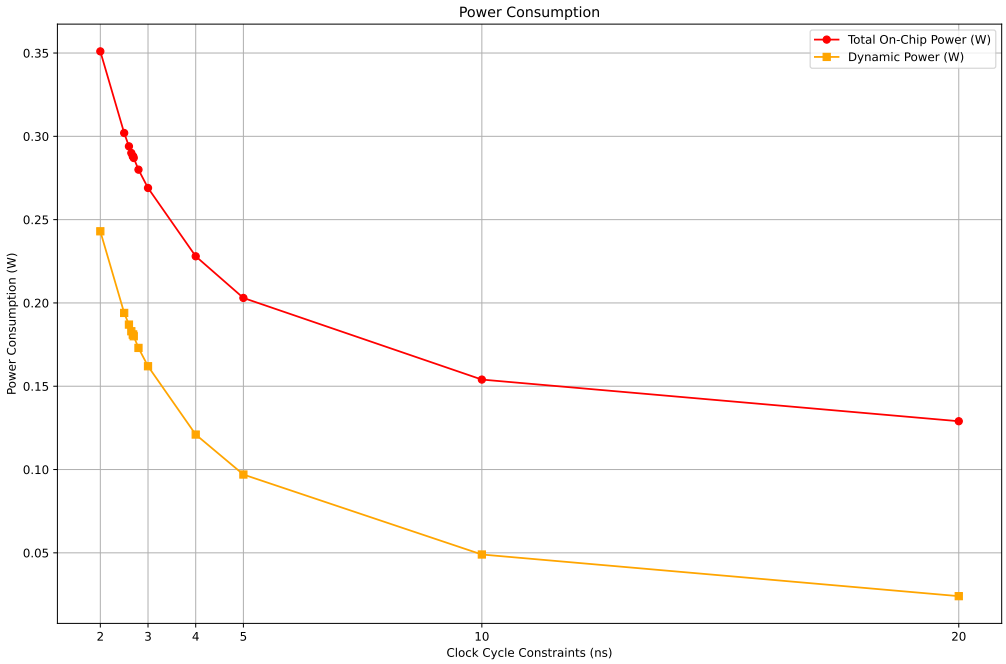

Figure 5.3: Plot of power analysis.

Dynamic power is the main power dissipation mechanism in digital circuits. Figure 5.4 on the left shows the distribution of on-chip power between dynamic and device static power, while the right chart breaks down the dynamic power into its components: clocks, signals, logic, and I/O at the clock cycle constraint of 2.69 ns. Dynamic power consumption in an FPGA design is influenced by various components of the circuit. Each part of the circuit contributes differently to the overall dynamic power, depending on its activity and utilization. I/O contributes the majority, around 81% of the total dynamic power. This suggests that data communication

with off-chip components or peripherals is the most power-hungry aspect of the design. Clocks contribute 14% of the dynamic power, this shows that the clocking circuitry is the second-largest dynamic power consumer.

On the contrast, the static power remains relatively constant, with slight variations. Static power is less influenced by the clock cycle constraints and is mainly determined by the leakage currents in the design.

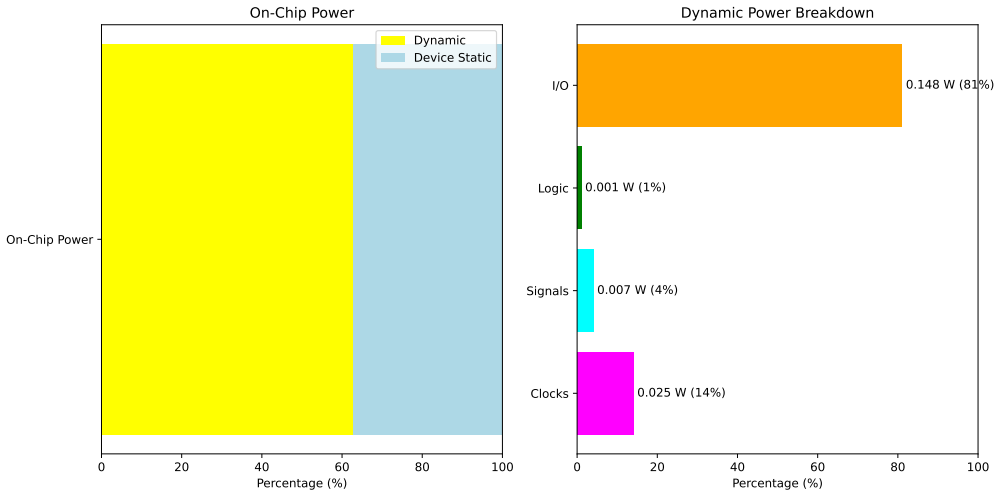

Figure 5.4: Power analysis break down.

Comparing the power report with the utilization report, it can be noticed that the design becomes more area-efficient at higher clock cycle constraints, as evidenced by the reduced number of Slice LUTs at 10 ns and 20 ns constraints. Both total and dynamic power consumption decreases significantly with relaxed timing constraints, indicating improved power efficiency.

6

Conclusion

In this thesis, the control interface between RISC-V processors and NoC in a DNN accelerator IP was implemented using AXI-Lite in VHDL. It can transmit data by reading or writing to certain addresses at a throughput of 100%, providing sufficient bandwidth for a high-speed command controlling data interface while maintaining low energy and hardware resources costs.

In this thesis, we first proposed several alternatives for the given task and chose to implement AXI4-Lite for its high performance over other solutions. Then we designed each block separately to fit in the pre-existing system. The design was simulated in Modelsim and checked with golden test vectors to ensure function correctness. The synthesis in Vivado was run on a Xilinx ZYNQ 7020 FPGA board and we also evaluated the resource utilization and energy consumption of the design.

Results showed that the design was power and resource-efficient. During the testing stage of the project, Imsys unfortunately went bankrupt, resulting in the unavailability of the necessary FPGA for testing. The integration of our design with the system was partly done. Additionally, we lacked licenses for advanced FPGA boards in Vivado to synthesize the design; the board we had did not possess enough I/O ports to take onboard testing. Due to the bankruptcy, the engineer responsible for the NoC component was on leave and could not provide support for the integration test. We managed to simulate and synthesize our design with the NoC but there is no guarantee they work perfectly together as we have limited resources to carry through experiments, which could be a future work for the project.

Bibliography

[1] V. Sze, Y.-H. Chen, T.-J. Yang, and J. S. Emer, "Efficient processing of deep neural networks: A tutorial and survey," *Proceedings of the IEEE*, vol. 105, no. 12, pp. 2295–2329, 2017.

[2] J. Faraone, M. Kumm, M. Hardieck, P. Zipf, X. Liu, D. Boland, and P. H. Leong, "Addnet: Deep neural networks using FPGA-optimized multipliers," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 28, no. 1, pp. 115–128, 2019.

[3] R. Raina, A. Madhavan, and A. Y. Ng, "Large-scale deep unsupervised learning using graphics processors," in *Proceedings of the 26th annual international conference on machine learning*, 2009, pp. 873–880.

[4] H. S. Lee and J. Wook Jeon, "Accelerating deep neural networks using FPGAs and ZYNQ," in *2021 IEEE Region 10 Symposium (TENSYMP)*, 2021, pp. 1–4.

[5] *AMBA AXI Protocol Specification*, Arm, 2010. [Online]. Available: https://documentation-service.arm.com/static/5f915971f86e16515cdc34a6?token=

[6] C. Zhang, P. Li, G. Sun, Y. Guan, B. Xiao, and J. Cong, "Optimizing FPGA-based accelerator design for deep convolutional neural networks," in *Proceedings of the 2015 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*, ser. FPGA '15. New York, NY, USA: Association for Computing Machinery, 2015, p. 161–170. [Online]. Available: https://doi.org/10.1145/2684746.2689060

[7] P. N. Whatmough, S. K. Lee, D. Brooks, and G.-Y. Wei, "DNN engine: A 28-nm timing-error tolerant sparse deep neural network processor for IoT applications," *IEEE Journal of Solid-State Circuits*, vol. 53, no. 9, pp. 2722–2731, 2018.

[8] M. Pagani, E. Rossi, A. Biondi, M. Marinoni, G. Lipari, and G. Buttazzo, "A Bandwidth Reservation Mechanism for AXI-based Hardware Accelerators on FPGAs," in *31st Euromicro Conference on Real-Time Systems (ECRTS 2019)*, Stuttgart, Germany, Jul. 2019. [Online]. Available: https://hal.science/hal-02407835

[9] *SmartConnect v1.0 LogiCORE IP Product Guide*, AMD, 2022. [Online]. Available: https://docs.xilinx.com/r/en-US/pg247-smartconnect/SmartConnect-v1.0-LogiCORE-IP-Product-Guide

[10] F. Restuccia, A. Biondi, M. Marinoni, G. Cicero, and G. Buttazzo, "AXI hyperconnect: A predictable, hypervisor-level interconnect for hardware acceler-

ators in FPGA soc," in *2020 57th ACM/IEEE Design Automation Conference (DAC)*, 2020, pp. 1–6.

[11] (2024) Hw demonstrator for all axi accelerator interface. Imsys AB.

[12] P. A. Simpson, *FPGA design*, 2nd ed., Springer, Ed., 2016.

[13] O. Mencer, D. Allison, E. Blatt, M. Cummings, M. J. Flynn, J. Harris, C. Hewitt, Q. Jacobson, M. Lavasani, M. Moazami, H. Murray, M. Nikravesh, A. Nowatzyk, M. Shand, and S. Shirazi, "The history, status, and future of fpgas." *Communications of the ACM*, vol. 63, no. 10, 2020.

[14] I. Tuomi, *The Future of Semiconductor Intellectual Property Architectural Blocks in Europe*, 1st ed., M. Bogdanowicz, Ed., 2009.

[15] *Introduction to AMBA AXI4*, Arm, 2020. [Online]. Available: https://developer.arm.com/-/media/Arm%20Developer%20Community/PDF/Learn%20the%20Architecture/102202_0100_01_Introduction_to_AMBA_AXI.pdf?revision=369ad681-f926-47b0-81be-42813d39e132

[16] ARM Limited, *AMBA Specification Reference Manual*, 1st ed., ARM Limited, Cambridge, UK, 2003, Available: https://developer.arm.com/documentation/ihi0022/b.

[17] ——, *AMBA Specification Reference Manual*, 1st ed., ARM Limited, Cambridge, UK, 2010, Available: https://developer.arm.com/documentation/ihi0022/latest.

[18] *AMBA AXI and ACE Protocol Specification*, Arm, 2011. [Online]. Available: http://www.gstitt.ece.ufl.edu/courses/fall15/eel4720_5721/labs/refs/AXI4_specification.pdf

[19] B. Jacob, D. Wang, and S. Ng, *Memory Systems: Cache, DRAM, Disk*. Elsevier Science, 2010. [Online]. Available: https://books.google.se/books?id=SrP3aWed-esC

[20] L. Benini and G. De Micheli, "Networks on chips: a new soc paradigm," *Computer*, vol. 35, no. 1, pp. 70–78, 2002.

[21] ARM Ltd., *AMBA Specification (Rev 2.0)*, ARM Ltd., 1999, accessed: 2024-05-29. [Online]. Available: https://developer.arm.com/documentation/ihi0011/bc/

[22] Xilinx, *Vivado Design Suite User Guide: Logic Simulation*, 2022nd ed., October 2022, uG900 (v2022.2). [Online]. Available: https://www.xilinx.com/support/documentation/sw_manuals/xilinx2022_2/ug900-vivado-logic-simulation.pdf

[23] Xilinx Inc., *Zynq-7000 SoC Data Sheet: Overview*, v1.12 ed., Xilinx Inc., San Jose, CA, 2021, Available: https://www.amd.com/content/dam/amd/en/documents/products/adaptive-socs-and-fpgas/soc/zynq-7000-product-brief.pdf.

[24] Dan Gisselquist. (2019) Skid buffers. [Online; accessed 27-September-2024]. [Online]. Available: https://zipcpu.com/blog/2019/05/22/skidbuffer.html

[25] J. Axelson, *Serial Port Complete: Programming and Circuits for RS-232 and RS-485 Links and Networks*, 1st ed. Lakeview Research, 1998.

[26] AMD Inc., *VCU118 Evaluation Kit Datasheet*, AMD Inc., San Jose, CA, 2022, [Online; accessed 27-September-2024]. [Online]. Available: https://docs.amd.com/v/u/en-US/ug1224-vcu118-eval-bd

[27] *AXI Interconnect Product Guide*, Xilinx Inc., April 2014. [Online]. Available: https://www.xilinx.com/support/documentation/ip_documentation/axi_interconnect/v2_1/pg059-axi-interconnect.pdf