RWU Hochschule Ravensburg-Weingarten University of Applied Sciences

PROJECT REPORT

System On Chip

Guided by:

Prof. Dr. ing Andreas siggelkow

By:
Abhishek Abhishek - 36136 (Master-EMM)
Vikram Rudraraju - 36139 (Master-EMM)

January 26, 2023

Contents

List of Figures

1 General

A System on a Chip (SoC) is an integrated circuit (IC) that incorporates all computer or electronic system components into a single chip. It combines the functions of a central processing unit (CPU), memory, input/output interfaces, and other components onto a single chip. This allows for smaller, more efficient devices as all the components are interconnected and communicate on the same chip. SoCs are commonly used in mobile devices, embedded systems, and Internet of Things (IoT) devices. They are also being used in more traditional computer and networking applications as they can offer a cost-effective and space-saving alternative to conventional discrete component systems.

This project aims to add a peripheral to the ARM core on the Zybo board using the Vivado software. The peripheral comprises two 32-bit registers that are readable and writeable. The first register is connected to LEDs, which allows the user to control the state of the LEDs by writing data to the register. The second register is connected to switches/buttons, which enables the user to read the input from the switches/buttons by reading the data from the register. The connection between the ARM and the registers is through the AXI bus.[2]

The system is implemented using VHDL, and the software is programmed in C. The Vivado IP Integrator connects the registers to the AXI bus, enabling communication between the ARM and the registers. The SDK software provided by Xilinx is used to write the C software for the ARM core and test the system. The C software allows the user to read and write data to the registers, turn on and off the LEDs, and read the input from the switches/buttons.

The system is tested using a simple scan test that verifies the functionality of the system. The test demonstrates the ability to read and write data to the registers, turn on and off the LEDs, and read the input from the switches/buttons. The system is optimised, and bugs are fixed during testing to ensure it works correctly. The bitstream file of the system is generated using the Vivado software and loaded into the Zybo board. The system is tested again on the Zybo board to ensure it works correctly.

The final version of the system is delivered to the customer, including the VHDL code, C software, and the bitstream file. The customer can use this system in various embedded systems to control the LEDs and read the input from the switches/buttons on the Zybo board. This project provides an efficient way to control the LEDs and read the input from the switches/buttons on the Zybo board, which can be used in various embedded systems such as IoT devices, automation systems, and more.

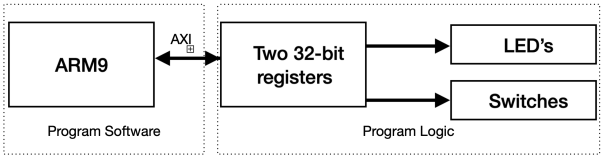

Figure 1: System Block Diagram

2 Requirement and Specification

2.1 Functional Requirements

Functional requirements are the specific features and capabilities that a system must have to meet its intended purpose. In the case of this project, the following applicable requirements are necessary for the system to function correctly.

1. The system should be able to read and write data to the two 32-bit registers: This requirement is essential for the system to function correctly. The registers are the main component of the peripheral, and the ability to read and write data to them is necessary for the system to control the LEDs and read the input from the switches/buttons.

2. The first register should be connected to LEDs, and the system should be able to turn on and off the LEDs by writing data to the register. This requirement is necessary for the system to control the LEDs. The LEDs are connected to the first register, and the system should be able to write data to turn the LEDs on and off.

3. The second register should be connected to switches/buttons, and the system should be able to read the input from the switches/buttons by reading data from the register. This requirement is necessary for the system to read the input from the switches/buttons. The switches/buttons are connected to the second register, and the system should be able to read data from the register to get input from the switches/buttons.

4. The system should be able to communicate between the ARM and the registers through the AXI bus: This requirement is necessary for the system to communicate between the ARM and the registers. The AXI bus transfers data between the ARM and the registers, and the system should be able to read and write data to the registers.

5. The system should be able to demonstrate a simple scan test that verifies the functionality of the system.

2.2 Non-Functional Requirements

Non-functional requirements are the characteristics or attributes of a system that do not directly contribute to its functionality but rather describe how the system should operate or perform. In the case of this project, the following non-functional requirements are necessary for the system to be effective and efficient.

1. The system should be implemented using VHDL, and the software should be programmed in C. This requirement is necessary for the system to be implemented using the appropriate technology. VHDL is used to design the system's hardware, and C is used to program the software for the ARM core.

2. The system should be designed to be integrated into the Zybo board using the Vivado software. This requirement is necessary for the system to be integrated into the Zybo board. The system should be designed to be integrated into the Zybo board using the Vivado software, which Xilinx provides.

3. The system should be designed to be compatible with the SDK software provided by Xilinx. This requirement is necessary for the system to be consistent with the SDK software provided by Xilinx. The system should be designed to work with the SDK software, which is used to write the C software for the ARM core and test the system.

4. The system should be designed to be easy to use and understand for the customer. This requirement is necessary for the system to be user-friendly. The system should be designed to be easy to use and understand for the customer so that it can be used without any problems.

5. The system should be designed to be robust and reliable for use in various embedded systems. This requirement is necessary for the system to be strong and reliable. The system should be robust and reliable for use in various embedded systems to be used in different applications without any problems.

2.3 Hardware Requirements

The following hardware devices are required for the system.

2.3.1 Zybo Board

The Zybo board is a development board used in our project as the platform for the system. It includes a Xilinx Zynq-7000 FPGA, a dual-core ARM Cortex-A9 processor, and various peripheral interfaces, making it a powerful and versatile platform for embedded systems. The FPGA can be configured to implement the peripheral for our project, while the processor is used to run the software that controls the peripheral. The peripheral interfaces of the Zybo board can connect to external devices, and the DDR3 memory can be used to store data and instructions for the processor. The onboard clock oscillator and power supplies provide the system with a clock signal and power. Additionally, the JTAG programming interface allows for easy programming and debugging of the system. Overall, the Zybo board is ideal for our project as it provides the necessary resources and functionality to implement the system efficiently and effectively.[1]

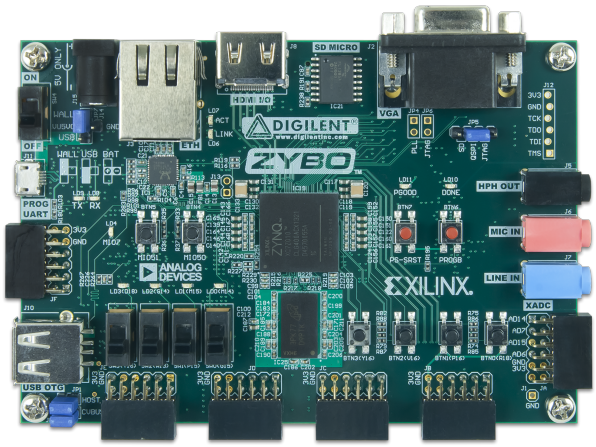

Figure 2: Zybo Board

2.3.2 LEDs

The LEDs used in our project are connected to the first register of the peripheral. They indicate the system's state and provide visual feedback to the user. The LEDs can be turned on and off by writing data to the first register, and they can be used to indicate different states of input, such as power on or off. The LEDs are a vital component of the system as they allow the user to understand its current state quickly.

2.3.3 Switches/buttons

These devices are connected to the second peripheral register and are used to provide input to the system. The switches/buttons are read by reading data from the second register, and they can be used to control the system or provide input for the system.

These hardware devices are necessary for the system to function correctly. Without the Zybo board, there would be no platform for the system, and without the LEDs and switches/buttons, there would be no way to control the system or provide input. The Zybo board is the core element that offers the ability to implement the peripheral, and the LEDs and switches/buttons are the devices that provide the user interface to the system.

2.4 Software Requirement

The following software tools are required for the project.

2.4.1 Vivado

This software designs the system's hardware using VHDL. It is used to create the design, synthesise it, and generate the bitstream file of the system. Vivado provides a complete design flow for devel-

oping embedded systems that include FPGA, processors, and peripherals. The software consists of a graphical user interface that allows designers to create and debug designs and a command-line interface for design flow automation.

2.4.2 SDK

This software is provided by Xilinx and is used to write the C software for the ARM core and test the system. SDK provides tools and libraries for creating software for the ARM processor on the Zybo board. It includes a C compiler, an editor, a debugger, libraries and drives access to the board's peripheral interfaces on the board.

2.4.3 C Compiler

This software compiles the C software written for the ARM core. This software converts the C source code into machine code that the processor can execute.

Vivado and SDK are provided by Xilinx, which makes them well-suited for use with the Zybo board. They are designed to work together and offer a complete design flow for creating embedded systems. The C compiler is a third-party tool that can be used to compile the software for the ARM core, and it is widely used in the embedded systems industry.

3 Document Overview

3.1 Technical Terms

The following are some technical terms that are commonly used in the context of simulation and synthesis for a Zybo board:

System-on-Chip (SoC): An integrated circuit that integrates all computer or electronic system components into a single chip. The Zybo board is based on a Xilinx Zynq-7000 All Programmable SoC.

Vivado Design Suite: A software tool developed by Xilinx for designing and implementing digital logic designs on programmable devices such as the Zynq SoC. The Vivado Design Suite includes a simulator, synthesis tool and other tools.

Simulator: A tool that allows designers to simulate the behaviour of digital logic designs before they are implemented on real hardware. The Vivado Simulator is used to simulate designs on the Zybo board.

Synthesis: The process of converting a high-level description of a digital logic design into a lower-level, technology-specific description that can be implemented on a programmable device. The Vivado Synthesis tool is used to synthesise designs for the Zybo board.

Bitstream: A file that contains the configuration data for a programmable device such as an FPGA or SoC. The bitstream is loaded onto the device to configure its logic elements and I/O resources.

Board Support Package (BSP): A software package that provides drivers and other software components needed to interact with the peripheral components of a development board. The BSP for the Zybo board is provided by Xilinx and is used to develop software for the board's SoC.

IP cores: pre-built digital logic components that can be used as building blocks for custom designs. The IP cores are provided by Xilinx and can be used with the Zybo board to create custom designs.

3.2 Naming Conventions

The following are some common naming conventions used for simulation and synthesis for a Zybo board:

Design project: A design project is typically created within the Vivado Design Suite and contains all the files and settings associated with a specific design. A design project is usually named according to the design it represents, for example, "zybo_led_blink".

Design files: Design files, such as Verilog or VHDL source files, are usually named according to the function or module they represent, for example, "led_blink.v".

Simulation files: Simulation files, such as testbench files, are usually named according to the design file they are associated with, for example, "led_blink_tb.v".

Synthesis files: Synthesis files, such as constraint files, are usually named according to the function or module they constrain, for example, "led_blink_constraints.xdc".

IP cores are pre-built digital logic components that can be used as building blocks for custom designs. They are usually named according to their function, for example, "axi_gpio".

Software files: Software files, such as drivers and application code, are usually named according to their function, for example, "led_blink_app.c"

It is worth noting that these are conventions and not strict rules, and some variations may be used depending on the specific project. It is always best to check with the project-specific guidelines or documentation for the naming conventions.

4 Product Overview

The Chapter deals with the system block diagram and the functional block diagram of the project.

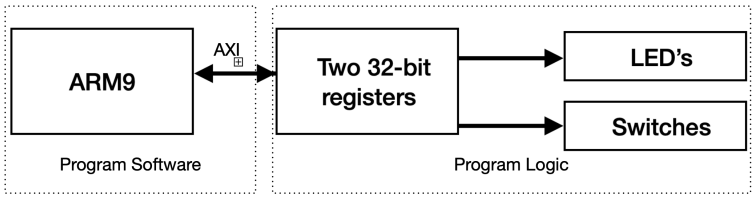

Figure 3: Top Level View of The System

The above block diagram gives a complete overview of the system. The system is divided into two major parts. The program software contains the processor; we used an ARM9 processor in this project. The Program logic consists of two 32-bit registers and 4-LEDs 4-Switches. The communication between Program software and program logic is established using the AXI bus controller. The AXI acts as a bridge between these two system units.

A block diagram for a Zybo board connected with an AXI (Advanced Extensible Interface) bus would typically show the main components of the board and how they are connected through the AXI bus.

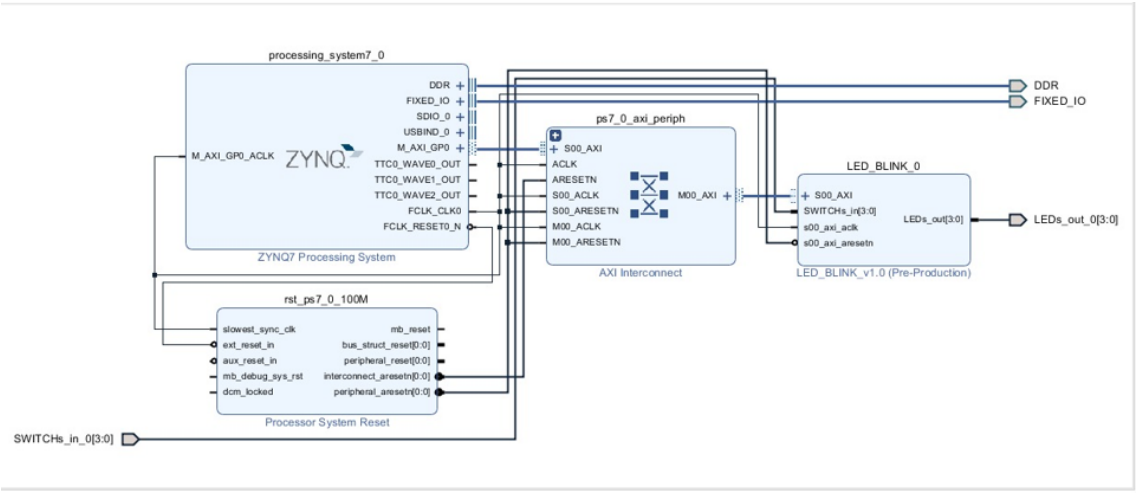

Figure 4: Zybo Board Block diagram Connected with AXI Bus

The main components would include:

Processor: The Zynq-7000 All Programmable SoC processor is typically an ARM Cortex-A9 dual-core processor.

Programmable Logic (PL): This is the programmable logic of the Zynq-7000 SoC, which can be configured to implement custom digital logic designs.

AXI Bus: This interface connects the processor and the programmable logic with the peripheral components. The AXI bus transfers data between these components, allowing the processor to communicate with and control the peripherals.

Peripherals: These are the various peripheral components that are available on the Zybo board, such as LEDs, buttons, USB, Ethernet, and HDMI interfaces. These peripherals are connected to the AXI bus, allowing the processor to communicate with and control them.

In summary, the Zybo board comprises a Processor, PL and Peripherals interconnected by an AXI bus, allowing data to be transferred and shared between the different components.

5 Architecture Concepts

5.1 AXI bus Development

The Advanced eXtensible Interface (AXI) bus is a high-performance, low-power interface used to transfer data between different components in a system. In this project, the AXI bus transfers data between the ARM core on the Z-Board FPGA and the two 32-bit registers that make up the peripheral.

The AXI bus is a parallel interface that consists of several signal lines for data transfer, address, and control. It is divided into three main channels.

1. The write channel transfers data from the ARM core to the peripheral. The ARM core writes data to the registers through this channel.

2. The read channel transfers data from the peripheral to the ARM core. The ARM core reads data from the registers through this channel.

3. The control channel transfers the control signals to the ARM core and the peripheral. The control signals include address, write enable, read enable, etc.

   The AXI bus is a flexible and scalable interface connecting various components in a system. It supports different transfer modes, such as burst and single transfer, and can transfer data at different speeds depending on the system's requirements. In this project, the AXI bus transfers data at a rate of up to 1 MHz, which is suitable for the system requirements.

   The AXI bus is a powerful interface that provides high-performance and low-power data transfer between the ARM core and the peripheral. It allows the ARM core to read and write data to the registers at high speed and the peripheral to send data to the ARM core at high speed, making it a crucial component of the system.

5.2 Write Process through AXI

The write channel in the AXI bus transfers data from the ARM core to the peripheral. Writing data to the peripheral is done through a handshake mechanism, a sequence of signals to ensure that the data transfer is done correctly. The handshake mechanism in the write channel consists of the following steps.

1. The ARM core sends a write address to the peripheral, indicating the location where the data will be written.

2. The peripheral sends an acknowledge signal (AWVALID) back to the ARM core, which indicates that it has received the write address.

3. The ARM core sends the data to be written along with a write enable signal (WVALID) to the peripheral.

4. The peripheral receives the data and write enable signal and sends a write response signal (BVALID) back to the ARM core.

5. The ARM core receives the write response signal and sends a write-done signal (WLAST) to the peripheral, which indicates that the writing process is complete.

This handshake mechanism ensures that the data transfer is done correctly and that both the ARM core and the peripheral are aware of the status of the transfer. The handshake mechanism also ensures that the data is written to the correct location in the peripheral and that the peripheral is ready to receive the following data transfer. The write channel through the handshake mechanism is a reliable way to transfer data between the ARM core and the peripheral, ensuring the consistency of the data written.

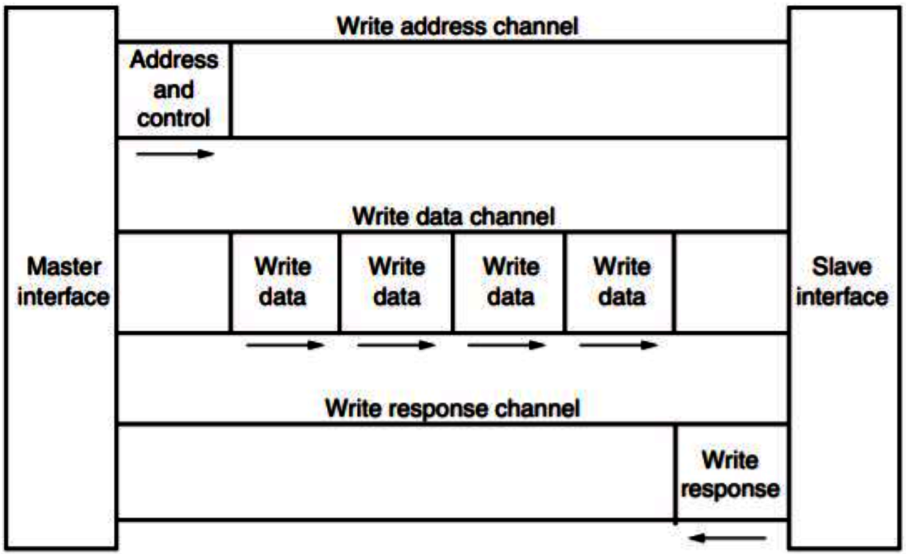

Figure 5: AXI Write Channel

5.3 Read Process through AXI

The read channel in the AXI bus transfers data from the peripheral to the ARM core. The process of reading data from the peripheral is done through a handshake mechanism, a sequence of signals used to ensure that the data transfer is done correctly. The handshake mechanism in the read channel consists of the following steps.

1. The ARM core sends a read address to the peripheral, indicating the location where the data will be read.

2. The peripheral sends an acknowledge signal (ARVALID) back to the ARM core, which indicates that it has received the read address.

3. The peripheral sends the data and a read response signal (RVALID) to the ARM core.

4. The ARM core receives the data and read response signal and sends a read-done signal (RLAST) to the peripheral, which indicates that the reading process is complete.

This handshake mechanism ensures that the data transfer is done correctly and that both the ARM core and the peripheral are aware of the status of the transfer. The handshake mechanism also ensures that the data is read from the correct location in the peripheral and that the peripheral is ready to receive the following data transfer. The read channel through the handshake mechanism is a reliable way to transfer data between the peripheral and the ARM core, ensuring the consistency of the data read.

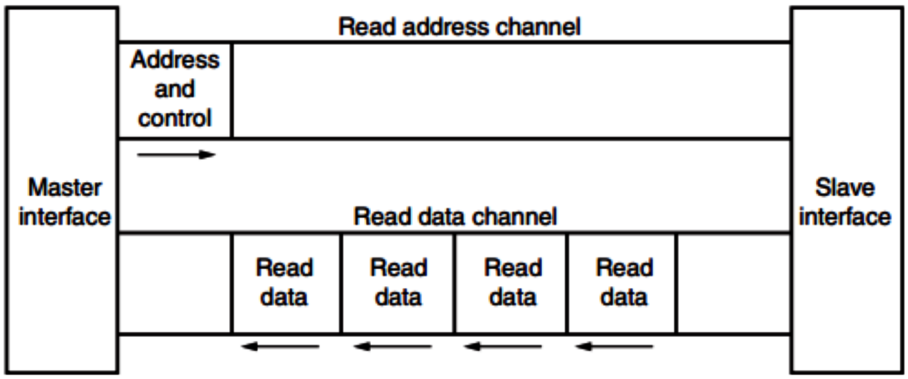

Figure 6: AXI Read Channel

5.4 AXI Transaction Process

Transactions are a fundamental concept in the AXI bus protocol, and they are used to transfer data between the master (ARM core) and the slave (peripheral). In the case of this project, the peripheral is the slave, and the ARM core is the master.

The above diagram shows the write burst transactions utilising the AXI bus to send the data to the particular address.

For write transactions, the process is as follows.

1. The Master sends a written address to the slave through AWADDR signal, which indicates the location in the slave where the data will be written.

2. The slave sends an acknowledge signal (AWVALID) back to the Master, which indicates that it has received the write address.

3. Slave memory checks whether the address is writeable and sends confirmation signals AWREADY to Master.

4. Then Master sends the data (i.e, DATA(A0) to DATA(A2)) to be written through WDATA. Then slave writes the data and sends the write response signal to the Master.

5. The Master receives the write response signal and sends a write done signal (WLAST) to the slave, which indicates that the write process is complete.

In addition to the single transfer mode, AXI also supports burst transactions. A burst write transaction transfers a continuous block of data to the peripheral. For burst write transactions, the process is as follows

Here the ARM core is the Master, and the peripheral is the Slave.

1. The ARM core sends a write address to the peripheral, indicating the location where the data will be written.

2. The peripheral sends an acknowledge signal (AWVALID) back to the ARM core, which indicates that it has received the write address.

3. Slave memory checks whether the address is writeable and sends confirmation signals AWREADY to Master.

4. The ARM core sends the data (i.e., DATA(A0) to DATA(A2)) to be written through WDATA. Then the slave writes the data and sends the write response signal to the Master.

5. The ARM core sends more data to the peripheral until the burst is completed and sends a write-done signal (WLAST) to the peripheral, indicating that the write burst is complete.

The AXI bus allows for high-speed data transfer between the peripheral and the ARM core, and the transaction with the burst process provides an efficient way to transfer a large amount of data in one go. This is particularly useful when moving large data blocks or when the peripheral requires continuous data transfer.

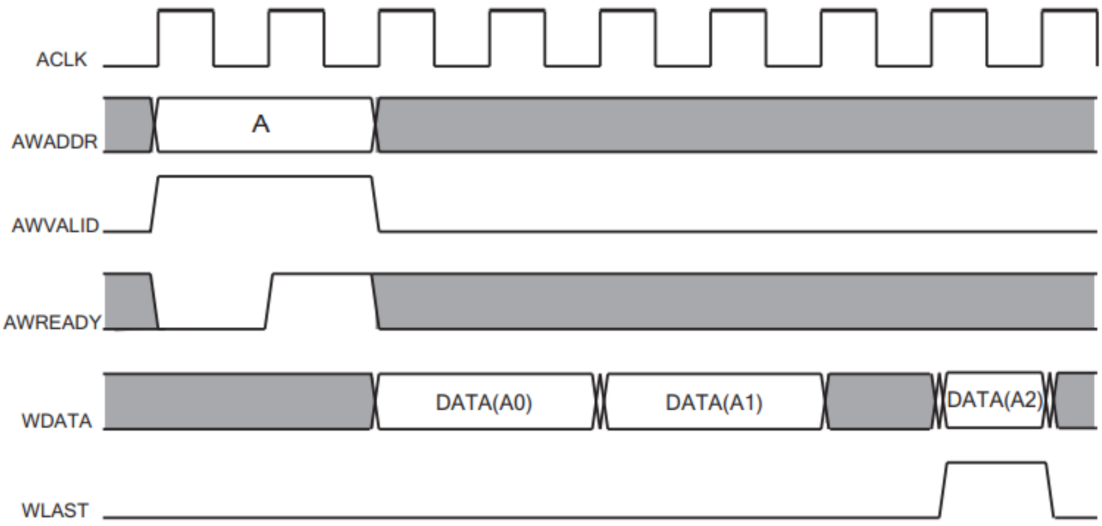

Figure 7: AXI Transaction Process

6 Description of the Design Elements

6.1 AXI Interface

The Advanced eXtensible Interface (AXI) bus is a high-performance, low-power interface used in this project to transfer data between the ARM core on the Z-Board FPGA and the two 32-bit registers that make up the peripheral. The AXI bus is a parallel interface that consists of several signal lines for data transfer, address, and control. It is divided into three main channels: the write channel, the read channel and the control channel. The write channel transfers data from the ARM core to the peripheral; the read channel transmits data from the peripheral to the ARM core, and the control channel transfers control signals between the ARM core and the peripheral. The AXI bus is a flexible and scalable interface connecting various components in a system. It supports different transfer modes, such as burst and single transfer, and can transfer data at different speeds depending on the system's requirements. In this project, the AXI bus transfers data at a rate of up to 1 MHz, which is suitable for the system requirements. The AXI bus is a powerful interface that provides high-performance and low-power data transfer between the ARM core and the peripheral. This interface allows the system to read and write data to the registers at high speed and the peripheral to send data to the ARM core at high speed, making it a crucial component of the system.

Figure 8: AXI Interface

6.2 Registers

The registers used in this project are two 32-bit registers, one for controlling LEDs and the other for receiving inputs from switches/buttons. These registers are connected to the ARM core through the AXI bus and transfer data between the peripheral and the ARM core.

1. LED control register: This register is used to control the state of the LEDs. The register is 32-bits wide, and each bit corresponds to one LED. Writing a '1' to a bit in the register turns on the corresponding LED, and writing a '0' turns it off. The address of this register is programmable, which means that it can be configured to any address that is not used by other peripherals in the system.

2. Input register: This register is used to read the state of the switches/buttons. The register is 32-bits wide, and each bit corresponds to one switch/button. A '1' in a bit of the register indicates that the corresponding switch/button is pressed, and a '0' indicates that it is not pressed. The address of this register is programmable, which means that it can be configured to any address that is not used by other peripherals in the system.

These registers are connected to the AXI bus, allowing the ARM core to read and write data. The AXI bus is a parallel interface, which allows for a high-speed data transfer between the registers and the ARM core. The data transfer between the registers and the ARM core is done through a handshake mechanism, which ensures that the data transfer is done correctly and that both the ARM core and the peripheral are aware of the transfer status.

7 Testing and Debugging

7.1 Behavioural Simulation

Behavioral simulation tests the behaviour of a digital circuit before it is implemented in hardware. This is done by creating a test bench file, a VHDL file that simulates the inputs and outputs of the circuit. The test bench file tests the circuit's functionality and verifies that it meets the specifications.

In this project, a test bench file is used to simulate the behaviour of the peripheral and the AXI bus. The test bench file is used to test the functionality of the peripheral, including the LED control register and the input register, and to verify that they meet the specifications. The test bench file is also used to simulate the behaviour of the AXI bus, including the handshake mechanism, and to verify that it meets the specifications.

The test bench file is helpful in this project because it allows for the functional verification of the system before it is implemented in hardware. This helps to identify and correct any issues with the system before it is implemented, which saves time and resources. The test bench file can also be used to test different scenarios, such as input conditions and system states, which helps ensure the system is robust and reliable.

The test bench file also allows for automated and regression testing, which can be used to verify the system's functionality over time. The test bench file can be integrated with a version control system, which allows tracking the changes and testing of the system's functionality at different project stages.

Overall, using a test bench file in this project is beneficial as it helps ensure that the system meets the specifications and is robust, reliable and easy to maintain.

Figure 9: Behavioural Simulation

7.2 Synthesis

This project synthesises converting the VHDL code into a gate-level description that can be implemented on the FPGA. This process is done using a synthesis tool, such as Vivado, used in this project.

The first step in the synthesis process is to create a new project in Vivado and import the VHDL files that make up the system, including the peripheral and the AXI bus. These files are then analysed and elaborated, which means that the VHDL code is checked for syntax errors, and the design hierarchy is established. Afterwards, the design is optimised and technology-mapped to the specific FPGA device used in this project. This process includes logic synthesis, converting the VHDL code into a gate-level description, and place and route, mapping the design to the physical resources of the FPGA.

The final step in the synthesis process is to generate the bitstream file, which is the file that is loaded onto the FPGA to configure it. The bitstream file contains all the information needed to configure the FPGA, including the gate-level description of the design, the interconnections between the gates, and the configuration of the I/O resources.

The synthesis process is an essential step in the design flow of this project. It allows the system to be implemented in hardware, and the bitstream file is loaded onto the FPGA to configure it. This process is done efficiently and correctly, which ensures that the system meets the specifications and is robust, reliable, and easy to maintain.

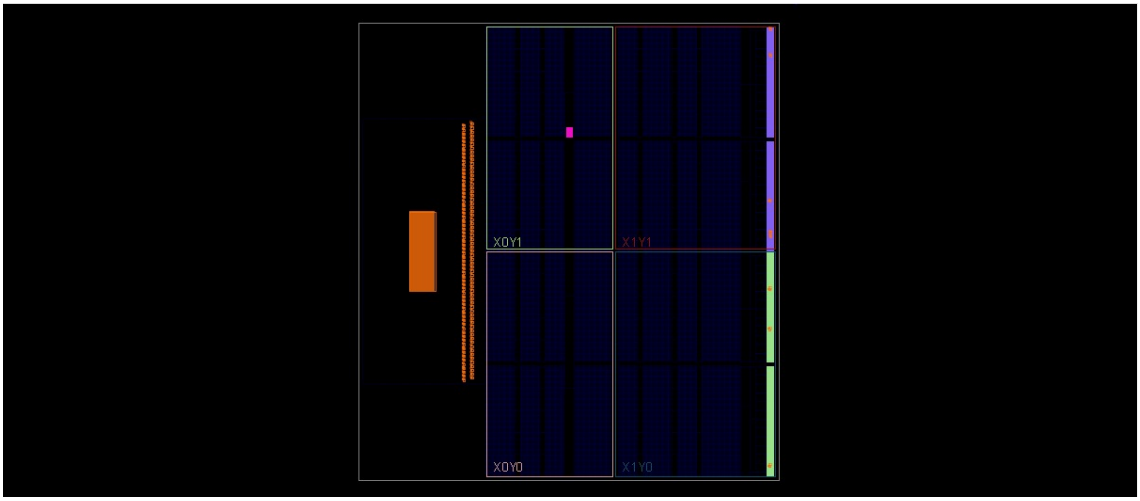

Figure 10: Synthesis

8 Conclusion

This project aimed to add a peripheral to the ARM core on the Z-Board FPGA. The peripheral consisted of two 32-bit registers, one for controlling LEDs and the other for receiving inputs from switches/buttons. The connection between the ARM and the register was the AXI bus, which allows for high-performance and low-power data transfer.

The system was implemented in VHDL, and the software was programmed in C. A simple scan test was demonstrated to show the functionality of the system. The system was verified through behavioural simulation using a test bench file, and the system was synthesised and implemented using the Vivado software. The project demonstrates the use of the AXI bus and how it can connect a peripheral to an ARM core. The project also illustrates how a simple peripheral can be implemented on an FPGA and how an ARM core can control it. The project is an excellent example of using the AXI bus and implementing a simple peripheral on an FPGA.

Overall, this project successfully achieved its goal of adding a peripheral to the ARM core on the Z-Board FPGA. The system was implemented, tested, verified and met the specifications and requirements. The project provides an excellent example of using the AXI bus and implementing a simple peripheral on an FPGA. It can be used as a reference for future projects that involve using the AXI bus and the implementation of peripherals on FPGAs.

References

[1] Zybo fpga board reference manual, revised february 27, 2017.

[2] Martin A. Enderwitz Robert W Stewart Louise H. Crockett, Ross A. Elliot. The zynq book embedded processing with the arm cortex-a9 on the xilinx zynq-7000 all programmable soc. 20(10):3782–3795, 2019.