KLE TECH KLE Technological University Creating Value Leveraging Knowledge

School of
Electrical and Electronics Engineering

Mini Project Report

on

RISC V Based SoC Design-SPI SPI Protocol

By:

1. **Rahul G Teli** USN: 01FE21BEE008
2. **Malhar Kulkarni** USN: 01FE21BEE016
3. **Chandru Thomare** USN: 01FE21BEE022
4. **Chandrashekhar Angadi** USN: 01FE21BEE031

Semester: V, 2023-2024

Under the Guidance of

Dr. Saroja Siddamal

K.L.E SOCIETY'S

KLE Technological University, HUBBALLI-580031 2023-2024

SCHOOL OF ELECTRICAL AND ELECTRONICS ENGINEERING

CERTIFICATE

This is to certify that project entitled **"RISC V Based SoC Design-SPI"** is a bonafide work carried out by the student team of" **Rahul G Teli- 01FE21BEE008, Malhar Kulkarni-01FE21BEE016, Chandru Thomare - 01FE21BEE022, Chandrashekar R Angadi - 01FE21BEE031".** The project report has been approved as it satisfies the requirements with respect to the minor project work prescribed by the university curriculum for BE (V Semester) in School of Electrical and Electronics Engineering of KLE Technological University for the academic year 2023-2024.

Dr. Saroja Siddamal	**Dr. A.B. Raju**	**Dr. Basavaraj S Anami**
Guide	**Head of School**	**Registrar**

External Viva:

Name of Examiners	**Signature with date**
1.	
2.	

ACKNOWLEDGMENT

We would like to express our sincere gratitude to KLE Technological University, Hubballi for providing us the facilities and resources to carry out our project on RISC-V Based SoC Design with SPI interface. We are especially thankful to the Head of School of Electrical and Electronics, Dr. A.B. Raju, for her constant motivation and guidance throughout this endeavour. We would also like to thank our project guide Dr. Saroja V. Siddamal for her invaluable insights, feedback, and direction at every stage of this project. Her vast knowledge and experience in this field steered us in the right direction whenever we faced any difficulties. In addition, we are grateful to the entire VLSI Design faculty team for clarifying our doubts patiently and helping us gain a thorough understanding of advanced concepts related to our project. We also appreciate the lab technicians for promptly addressing all our queries on the software tools and ensuring smooth progress of simulations. Finally, we would like to express our heartfelt gratitude towards our parents and family members for their unconditional love, emotional support and encouragement that gave us the motivation to work diligently on this challenging yet fulfilling project. We dedicate our project work to them as without their blessings, none of this would have been possible. Once again, we thank everyone who was directly or indirectly involved in making our efforts come to fruition. This acknowledgment reaffirms our sincere appreciation

- Rahul G Teli, Malhar Kulkarni, Chandru Thomare, Chandrashekar R Angadi

ABSTRACT

As embedded systems grow more complex, effective communication between components becomes critical for overall system reliability and efficiency. The serial peripheral interface (SPI) has emerged as a popular bus protocol for achieving high-speed synchronized data transfer with reduced pin requirements. This project focuses on designing and integrating an SPI controller subsystem into a RISC-V based System-on-Chip (SoC). The SPI controller is implemented in Verilog HDL and seamlessly interfaces with the RISC-V processor core to enable communication with external peripherals. It supports essential features including full-duplex data transmission, programmable clock polarity and phase options, interrupt generation and arbitration logic for multi-master configurations. A detailed register set with configurable control bits and data buffers allows flexibility to connect to a wide range of SPI-based sensors, ADCs, displays as per application requirements. Comprehensive testing procedures in simulation validate error-free functionality for corner cases. The design is mapped to the ARTY-7 FPGA development board for prototype demonstration. The realized SoC allows the RISC-V processor to efficiently orchestrate data exchange with multiple external devices using the integrated SPI subsystem. This simplifies integration challenges in embedded product designs and serves as a vital building block for realizing more complex SoC architectures. The modular Verilog implementation, register specifications and test methodology documented in this project will facilitate future enhancements as well as integration of the design into variants of RISC-V based SoCs

Contents

List of Tables

List of Figures

Chapter 1

Introduction

The Serial Peripheral Interface (SPI), a protocol designed for the efficient, synchronous communication between devices, is fundamental in the realm of embedded systems. Originating in the mid-1980s by Motorola for inter-chip communication, SPI has become a staple in facilitating dialogue between various slow-speed devices, including sensors, analog-to-digital converters, real-time clocks, and memory modules like flash memory. This protocol, characterized by its full-duplex communication ability, leverages a select line alongside separate clock and data connections to manage multiple devices, thereby providing an uninterrupted data transfer capability that supports up to 32 bits of simultaneous transmission.

1.1 Motivation

The urgency to monitor environmental quality, characterized by the analysis of gases and impurities both indoors and outdoors, necessitates innovative technological solutions. Traditional systems for detecting air pollution levels are often hampered by their lack of mobility and prohibitive costs. Herein lies the potential of System on Chip (SoC) technology, specifically those powered by the RISC V processor integrated with SPI, I2C, UART, and GPIO for sensor-based applications. SPI's full-duplex mode, alongside its capability for 32-bit data transfer and minimal power consumption due to its straightforward hardware configuration, underscores its advantage over alternatives like I2C.

1.2 Objectives

The primary goals of this endeavour are twofold:
To seamlessly integrate SPI protocol with the RISC V ET1032 Processor.
To conduct conclusive testing, verification, and validation of the SPI protocol within a RISC V based SoC, using the Arty-A7 FPGA board as a platform.

1.3 Literature survey

NXP-SPI Block Guide V03.06 by Motorola, INC.

The SPIF (SPI Interrupt Flag) bit is raised once a received data byte has been successfully transferred into the SPI Data Register. To clear this flag, one must read the SPISR register with SPIF set, followed by a subsequent read access to the SPI Data Register.
When SPIF is set to 1, it indicates that new data has been copied to SPIDR.
When SPIF is set to 0, it signifies that the transfer is not yet complete
The introduction provides an overview of the SPI (Serial Peripheral Interface) communication, covering its features and modes of operation. The master and slave wiring connections, involving MOSI (Master Out Slave In), MISO (Master in Slave Out), SCK (Clock Signal Line), and SS/CS (Slave Select/Chip Select), are detailed in the external signal description section. The memory map includes descriptions of important registers such as SPI Control Register-1, SPI Control Register-2, SPI Baud Rate Register, SPI Status Register, and SPI Data Register.
The functional description encompasses master and slave modes, transmission formats, clock phase controls, and polarity controls. It specifically addresses both CPHA=0 and CPHA=1 transfer formats, SPI baud rate generation, and the functionality of the slave select output. The comprehensive coverage of these aspects in the functional description contributes to a thorough understanding of the SPI communication protocol. [4] likely refers to a citation or source providing additional details on the SPI protocol.

Design and Implementation of a High-Speed Serial Peripheral Interface
Serial Peripheral Interface (SPI) is a synchronous communication protocol that enables serial data transfer between a master device and one or more slave devices. This work presents the implementation details of a high-speed SPI Master/Slave design based on the SPI Block Guide by Motorola (V03.06).
The design process is methodically developed, starting from the preliminary specifications, and culminating in the final physical implementation. The complete design is mapped onto Xilinx Vitex 5 FPGAs using Verilog 2001.
SPI is a widely used serial protocol for low-to-medium speed data transfers within and between integrated circuits and peripheral devices. It allows a microcontroller to communicate with various external components such as DACs, ADCs, and EEPROMs [1].

IP Core of Serial Peripheral Interface (SPI) with AMBA APB Interface
This SPI design successfully achieves a maximum frequency of 16 MHz and full-duplex transfer of 8-bit serial data in master mode. The functionality is modelled in Modalism and synthesized using Quartus Lite 16. The RTL netlist viewer in Quartus Lite 16 shows the data flow between different submodules. This is a simple interface that can be easily connected to the I/O ports of a microcontroller and used to interface with an APB bus [5].

Implementation of SPI Protocol in FPGA
This paper presents an implementation of the Serial Peripheral Interface (SPI) protocol on a Field Programmable Gate Array (FPGA). State machine diagrams are utilized for implementing the SPI Master and SPI Slave components. The design is coded in VHDL (Very High-Speed Integrated Circuit Hardware Description Language). Upon receiving an acknowledgement, the received data is checked and the simulation results are displayed [7].

Design and Implementation of Serial Peripheral Interface Protocol Using Verilog
This study presents the design and implementation of a master and slave Serial Peripheral Interface (SPI) using Verilog HDL. SPI is a synchronous serial communication protocol that enables full-duplex data transfer. It has two communication modes: master and slave.

The entire design is simulated and synthesized using the Xilinx ISE 13.3.1 design suite. The master device generates the serial clock while individual slave selects lines enable the slave devices.

SPI was initially created by Motorola as a hardware/firmware communication protocol and subsequently adopted by other companies. It is sometimes referred to as a "four-wire" serial bus. Many microprocessor and microcontroller chips use the SPI bus - a simple 4-wire serial interface for communication between controllers and peripherals. While primarily intended for host-peripheral connections, SPI can also connect two processors.

In fully operational mode, SPI features a synchronous data link with a Master/Slave interface. It allows both single-master and multi-master configurations. SPI bus is typically used within a PCB to enable high-speed, efficient data transfer between different ICs [6].

Design and Analysis of Serial Peripheral Interface for Automotive Controller
Serial Peripheral Interface (SPI) is a popular communication protocol that enables serial data transfer between a master and slave device over a short distance. However, existing SPI implementations face issues with clock synchronization and poor speed performance.

To improve the interface performance, certain automotive-specific parameters need to be considered. This work presents the design, simulation, verification and optimization of an SPI based on automotive interface standards. It incorporates improved power efficiency and a faster interface.

The attained speed and area are significantly better than the present SPI design. Owing to the speed-power trade-off in VLSI design, the total power dissipation obtained is higher than the existing SPI architecture, but acceptable [2].

Design and Simulation of SPI Master / Slave Using Verilog HDL
The objective of this paper is to develop and simulate a SPI (Serial Peripheral Interface) master and slave using Verilog HDL. SPI is a widely used interface for connecting peripherals to microprocessors and to each other. While various Motorola microcontrollers support transfer block sizes between two to sixteen bits, most literature states that the interface is limited to eight- or sixteen-bit data transfers. However, the serial nature of the interface allows larger data transfers to be implemented easily using control signals.

SPI can be configured as either a master or slave protocol. As a master, it can control up to 32 separate SPI slaves. The transmit and receive register widths can be set to greater than sixteen bits. Using SPI, as many devices as there are pins on the microcontroller can be connected, with significantly higher communication rates between ICs. SPI enables full duplex communication [8].

Design and Verification Serial Peripheral Interface (SPI) Protocol for Low Power Applications
Serial Peripheral Interface (SPI) technology enables high-speed data communication between devices and was designed to replace parallel connections, eliminating the need to route parallel buses on PCBs. Motorola first identified the circuitry behind SPI in the late 1970s when connecting peripherals to its initial 68000-based microcontroller unit. SPI was subsequently adopted more widely in the industry due to its easy interfacing and high-speed facilitating data transfers.

Today, SPI has a strong presence in embedded systems - whether in System-on-Chip (SOC) processors, microcontrollers like PIC and AVR, or more advanced 32-bit processors using ARM/PowerPC/x86 cores. These chips typically integrate SPI controllers capable of master or slave mode operation. SPI is also sometimes used for communication purposes in FPGAs. With its ability to enable fast real-time data transfer, SPI now provides a widely-used means of interfacing peripherals across devices.

Moreover, Intel's Enhanced SPI bus (eSPI) aims to replace the low pin count (LPC) bus by allowing higher throughput, lower 1.8V operating voltages, and shared use of SPI flash devices. Whereas LPC prevented firmware hubbing, eSPI permits performance/cost tradeoffs via separation from the SPI bus to improve performance or pin sharing to reduce costs [3].

1.4 Problem statement

This project aims not only at the design and implementation of the SPI protocol but also at its integration with the RISC V processing architecture. This is supplemented by a rigorous phase of testing and validation to ascertain the efficacy of the implemented design.

1.5 Application in Societal Context

The application spectrum of the SPI protocol significantly intersects with societal benefit, particularly in enhancing the functionality and interactivity of digital storage (MMC and SD cards), conversion modules (ADCs and DACs), and communication interfaces (Ethernet, USB, USART). Its utilization in real-time monitoring devices such as temperature and pressure sensors, as well as in consumer electronics like touchscreens and camera lenses, illustrates SPI's omnipresence in modern technology infrastructure. This ubiquity, indicative of SPI's reliability and efficiency, solidifies its position as an instrumental asset in the development of responsive and adaptable embedded systems.

1.6 Organization of report

• Chapter 1: Introduction

It includes motivation towards the project, objectives of project, literature survey done towards the problem statement, defining the problem statement, and looking at the application in a societal context, followed by project planning.

• Chapter 2: System design

This chapter includes the High-level functional block diagram and description of the functional blocks.

• Chapter 3: Implementation details

This chapter contains detailed information about implementation. It also Includes specifications used to design and propose the final system architecture. It contains the Hardware Interfacing Description with in-depth details in Firmware, Hardware Description, and the Port Mapping. It also includes the detailed information about the Registers Description and Finite State Machine.

• Chapter 4: Results and Discussions

This chapter includes the conclusion and results of implementation.

- Chapter 5: Conclusion

This is the concluding chapter that includes project closure and epilogue along with- the future scope of our project.

Chapter 2

System Design of SPI

This chapter elucidates the block-level design of the Serial Peripheral Interface (SPI) within a System on Chip (SoC) architecture. The focal point of this design is the seamless integration of the Processor ET1032 with the SPI controller and SPI master to facilitate efficient data transfer to the SPI slave. The architecture underpins the transition of data handling from parallel to serial transmission, underscoring the system's capability to bridge the computational core with peripheral communication modules effectively.

2.1 Overview of the System Architecture

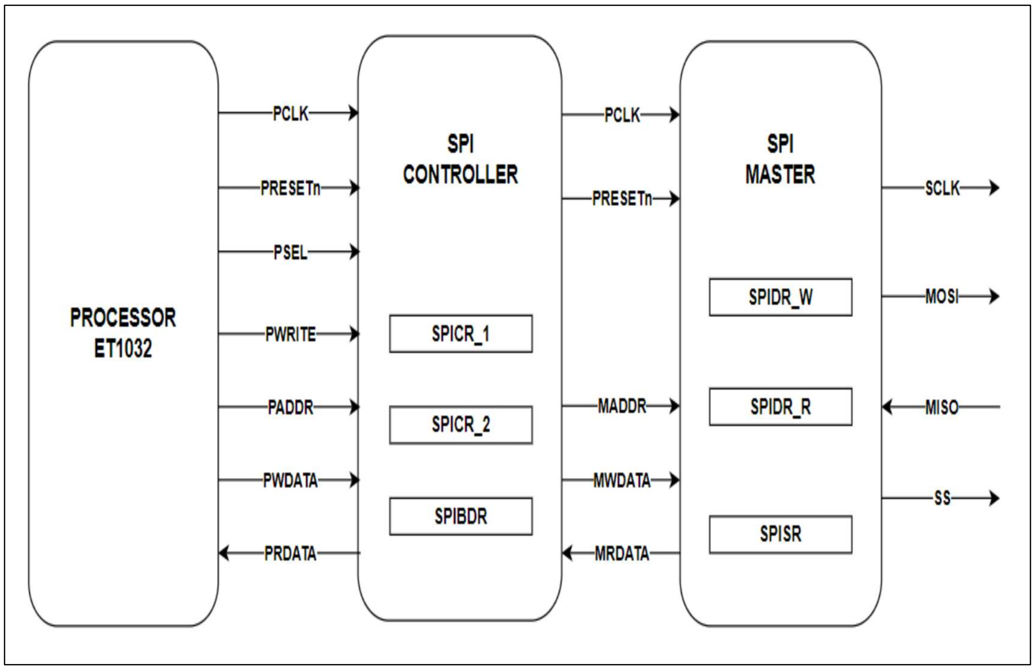


Figure 2.1: Block Diagram of SPI

The system's architecture is predicated on a tripartite structure comprising the Processor ET1032, the SPI Controller, and the SPI Master. At the outset, data is orchestrated in a parallel format from the Processor ET1032 to the SPI Controller, ensuring high-speed data handling within the core system. Subsequently, the SPI Controller interfaces with the SPI Master, maintaining the parallel data framework to uphold the integrity and speed of data processing. The culmination of this data journey is marked by the transition from parallel to serial transmission as the SPI Master communicates with the SPI slave, signifying the core-to-peripheral data exchange pivotal to SPI's operational paradigm.

ET1032-Processor

The Processor consists of boot memory from where it fetches the instruction pointed by the program counter (PC), decodes that instruction, and gives values for different variables based on the instruction. addrb and dinb are the main signals that are configured. These Processor variables relate to the SPI Controller block.

SPI Controller

SPI Controller receives values for PADDR and PWDATA from the Processor. Depending on the PADDR values, the two control registers and the baud rate registers are configured, thereby the read or write operations are decided. This block provides different values of frequencies of clock as input to the Master block based on the SPIBDR register bits.

SPI Master

The signals SCLK, MOSI and SS are the output from the master and MISO is the input to the master. The data register consists of input write data register, which is transferred to the slave serially through the MOSI line based on SCLK and the read data register receives the input through the MISO line based on SCLK.

Data Transfer Mechanism

Data transfer within the system initiates with the Processor ET1032 transmitting data in a parallel format to the SPI Controller, which, in turn, processes and transmits this data to the SPI Master while maintaining the parallel configuration. The SPI Master then converts this parallel data into a serial format, facilitating the transmission to the SPI slave. This structured approach to data handling, transitioning from parallel to serial transmission, epitomizes the system's design philosophy, optimizing for speed and efficiency within the embedded SoC environment.

Chapter 3

Implementation details

3.1 SPI Protocol Architecture

This chapter delves into the intricate details of implementing the Serial Peripheral Interface (SPI) protocol. It encompasses a comprehensive examination of the system's architecture, including the finite state machine (FSM) of the SPI Master, and culminates in the elucidation of the final system's architecture. The dual nature of the implementation—firmware and hardware—is meticulously explored, offering insights into the functional and control mechanisms underlying the SPI protocol within the System on Chip (SoC) framework.

3.2 SoC Integration Design

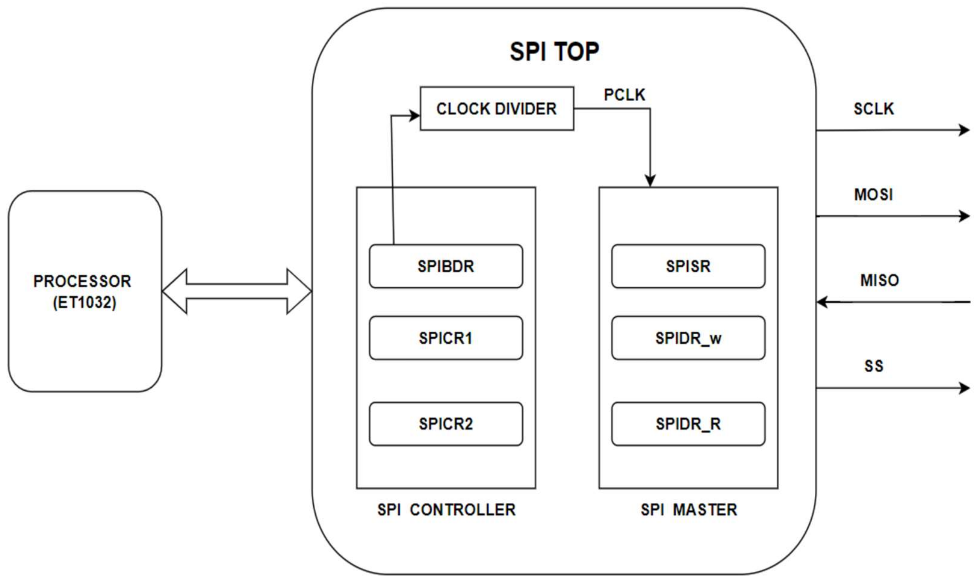
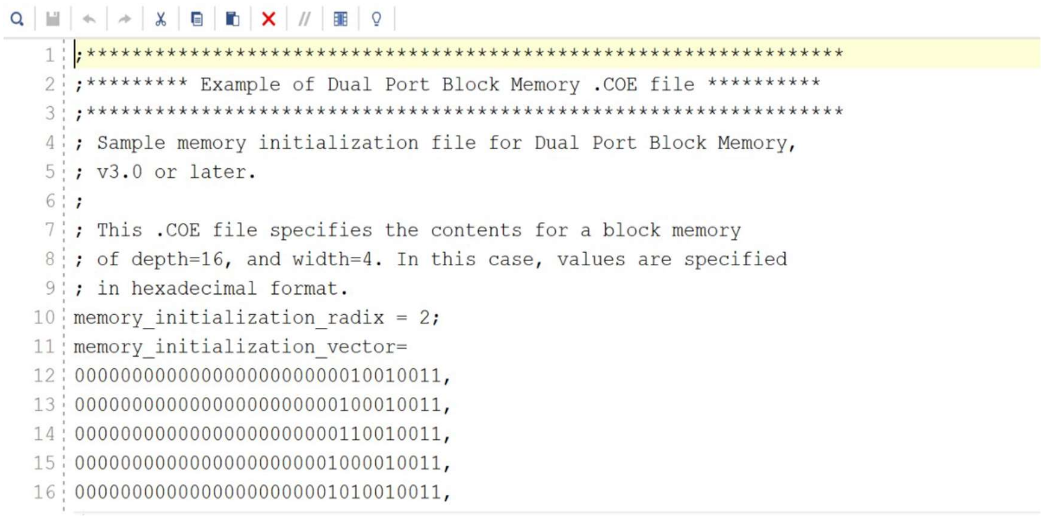


Figure 3.1: Architecture of SPI Protocol

The architecture of the SPI protocol is fundamentally designed to streamline the communication between the processor and peripheral devices. Figure 3.1 illustrates the structural overview, featuring both the Processor and the SPI top module. The interplay begins with the Processor dispatching clk_p, reset, and addrb signals to the SPI top module, which subsequently configures its registers accordingly. This configuration precipitates the serial data transfer from the master to the slave via the Master Out Slave In (MOSI) line, ensuring a seamless flow of information.

3.2.1 Firmware Implementation

Implemented within an Ubuntu environment, the firmware comprises four primary files: SPI.c, main.c, config.h, and SPI.h, collectively underpinning the SPI's operation.
SPI.c houses the core SPI program logic, mapping considerable functionality into the SPI operations. main.c is responsible for invoking functions defined within SPI.c, effectively serving as the entry point of the firmware. config.h specifies system-wide parameters, including the base address of the SPI (0x30000600). SPI.h declares essential variables correlating to SPI Control Register (SPI_CR), SPI Transmit Data Register (SPI_PWDATA), SPI Receive Data Register (SPI_PRDATA), and the SPI Status Register (SPI_SR), with designated addresses for efficient data handling and operational control.
Compilation is streamlined via a simple command (./build.sh) executed within the terminal, culminating in the generation of a soc_32.mif file containing binary instructions for the processor's boot memory.



```
;*******************************************************************
;********* Example of Dual Port Block Memory .COE file **********
;*******************************************************************
; Sample memory initialization file for Dual Port Block Memory,
; v3.0 or later.
;
; This .COE file specifies the contents for a block memory
; of depth=16, and width=4. In this case, values are specified
; in hexadecimal format.
memory_initialization_radix = 2;
memory_initialization_vector=
00000000000000000000000010010011,
00000000000000000000000100010011,
00000000000000000000000110010011,
00000000000000000000001000010011,
00000000000000000000001010010011,
```

Figure 3.2: Binary Instructions

3.2.2 Hardware Synthesis

The SoC hardware framework incorporates the RISC-V Processor, SPI Peripheral, memory locations, and a Clock Wizard for internal synchronization, illustrated in Figure 3.3. The Clock Wizard regulates instruction execution speeds, while the processor's modular architecture facilitates a wide spectrum of applications, from embedded devices to high-performance computing. The parallel existence of two memory blocks augments the system's capacity to manage data and instructions efficiently.

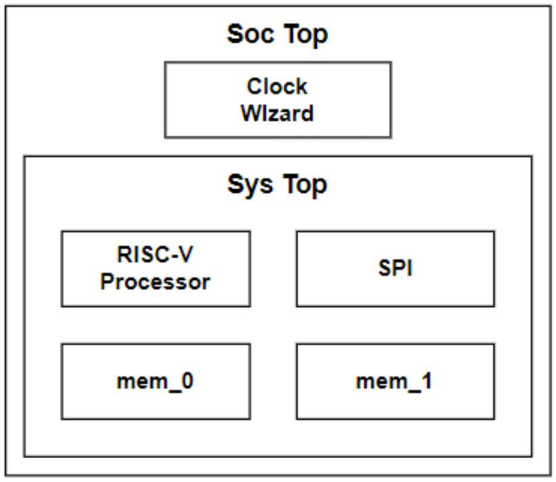


Figure 3.3: Block diagram of SoC

3.2.3 Port Mapping Considerations

Port mapping, illustrated in Figure 3.4, signifies the logical connection between the processor's control and data signals to those of the SPI, enabling coherent communication within the SoCs.

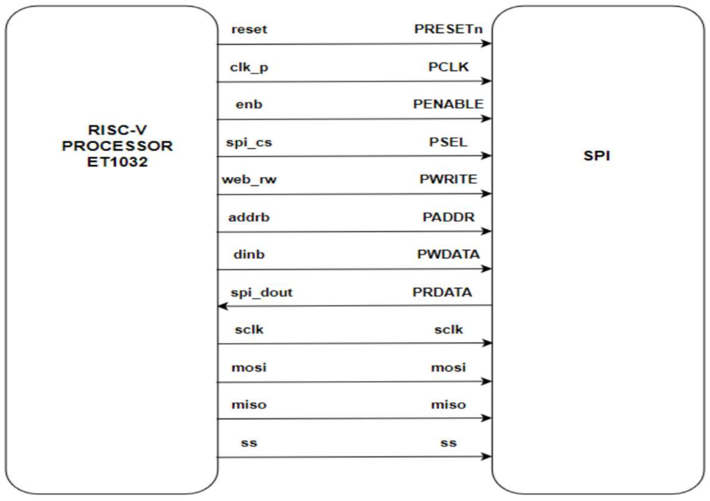


Figure 3.4: Port Mapping

3.3 Register Configurations and Descriptions

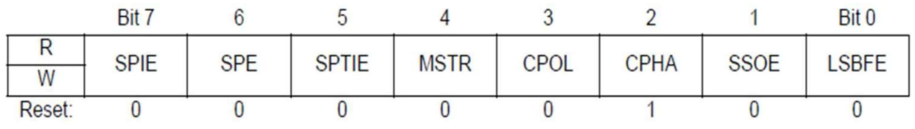
Integral to the SPI's functionality are several key registers: Control Register 1 (SPICR 1), Control Register 2 (SPICR 2), the Status Register (SPISR), and the Baud Rate Register.

SPICR 1 facilitates primary control operations, such as SPI enable/disable, master/slave mode selection, and clock polarity/phase settings.

SPICR 2 primarily deals with serial pin control distinctions between master and slave modes.
SPISR indicates transaction statuses, including interrupt flag, transmit empty flag, and mode fault occurrences.

The Baud Rate Register governs the SPI communication speed, adjustable through specific preselection and selection bits to match system requirements.

1) SPI Control Register 1 [8 bit] [Read/Write]:



	Bit 7	6	5	4	3	2	1	Bit 0
R / W	SPIE	SPE	SPTIE	MSTR	CPOL	CPHA	SSOE	LSBFE
Reset:	0	0	0	0	0	1	0	0

➢ SPIE — SPI Interrupt Enable Bit
This bit enables SPI interrupt requests, if SPIF or MODF status flag is set.
1 = SPI interrupts enabled.
0 = SPI interrupts disabled.

FUNCTIONS:

- The SPIE bit enables and disables all SPI interrupts. Setting SPIE to 1 allows interrupts from SPI transmission (TXIE) and reception (RXIE) to occur. Setting SPIE to 0 disables all SPI interrupts.

➢ SPE — SPI System Enable Bit
This bit enables the SPI system and dedicates the SPI port pins to SPI system functions.
1 = SPI enable, port pins are dedicated to SPI functions.
0 = SPI disabled (lower power consumption).

FUNCTIONS:

- The SPE (SPI Enable) bit enables and disables the SPI peripheral/module. Setting SPE to 1 enables the SPI peripheral so it can send and receive data. Setting SPE to 0 disables the SPI peripheral.

- SPTIE — SPI Transmit Interrupt Enable

  This bit enables SPI interrupt requests, if SPTEF flag is set.
  1 = SPTEF interrupt enabled
  0 = SPTEF interrupt disabled.

  FUNCTIONS:

- The SPTIE (SPI Transmit Interrupt Enable) bit enables and disables transmit interrupts in SPI. Setting SPTIE to 1 allows the SPI module to generate an interrupt when a transmission is complete. This signals the processor that transmission buffer is empty and new data can be written. Setting SPTIE to 0 disables transmit interrupts.

- MSTR — SPI Master/Slave Mode Select Bit

  This bit selects, if the SPI operates in master or slave mode. Switching the SPI from master to slave or vice versa forces the SPI system into idle state.
  1 = SPI is in Master mode.
  0 = SPI is in Slave mode.

  FUNCTIONS:

- The MSTR (Master/Slave Select) bit configures the SPI module as either a master or slave. Setting MSTR to 1 configures the SPI module as a master which generates the serial clock and controls the data flow. Setting MSTR to 0 configures the SPI module as a slave which receives the serial clock and data from a master SPI device.

- CPOL — SPI Clock Polarity Bit

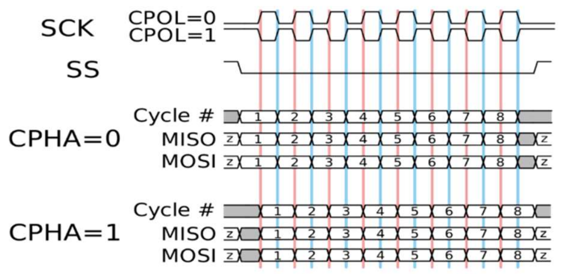
  This bit selects an inverted or non-inverted SPI clock. To transmit data between SPI modules, the SPI modules must have identical CPOL values.
  1 = Active-low clocks selected. In idle state SCK is high.
  0 = Active-high clocks selected. In idle state SCK is low.

  FUNCTIONS:

- The CPOL (Clock Polarity) bit sets the clock polarity for SPI communication. Setting CPOL to 0 configures the clock to be low when not transmitting data. This is the idle state. Setting CPOL to 1 configures the clock to be high when not transmitting data. The clock polarity needs to match the requirements of the SPI slave device.

➢ CPHA — SPI Clock Phase Bit

This bit is used to select the SPI clock format. In master mode, a change of this bit will abort a transmission in progress and force the SPI system into idle state.
1 = Sampling of data occurs at even edges (2,4, 6...16) of the SCK clock
0 = Sampling of data occurs at odd edges (1,3, 5...,15) of the SCK clock



Mode Of Operation

FUNCTIONS:

- The CPHA (Clock Phase) bit selects which clock edge data is sampled on. Setting CPHA to 0 configures the SPI module to sample data on the first edge of the clock period. This is useful when data is available at the beginning of the clock cycle. Setting CPHA to 1 configures sampling at the second edge of the clock period. This allows one clock cycle between shifting data and sampling. The phase needs to match the SPI slave device requirements.

➢ LSBFE — LSB-First Enable
This bit does not affect the position of the MSB and LSB in the data register. Reads and writes of the data register always have the MSB in bit 7.
1 = Data is transferred least significant bit first.
0 = Data is transferred most significant bit first.

FUNCTIONS:

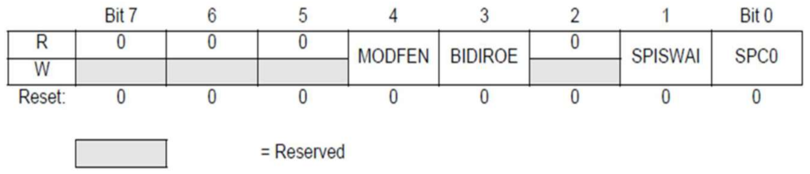
- The LSBFE (LSB First Enable) bit controls the bit order for SPI data transfers. Setting LSBFE to 1 configures the SPI module to send the least significant bit (LSB) first in the data transfer. This is the default and most common configuration. Setting LSBFE to 0 configures the SPI module to send the most significant bit (MSB) first. The bit order needs to match the requirements of the SPI slave device.

➢ SSOE — Slave Select Output Enable
The SS output feature is enabled only in master mode, if MODFEN is set

FUNCTIONS:

- The SSOE (Slave Select Output Enable) bit enables or disables the slave select (SS) pin output from the SPI module. Setting SSOE to 1 enables the SS output. This allows the SPI module to send the SS signal to an SPI slave device to select it before transferring data. Setting SSOE to 0 disables the SS output pin, preventing slave selection.

2) SPI Control Register 2 [8 bit] [Read/Write]:



	Bit 7	6	5	4	3	2	1	Bit 0
R	0	0	0	MODFEN	BIDIROE	0	SPISWAI	SPC0
W	▨	▨	▨			▨		
Reset:	0	0	0	0	0	0	0	0

▨ = Reserved

➢ The MODFEN bit enables or disables mode fault detection. Setting MODFEN to 1 allows detection of mode faults in master mode. Setting it to 0 disables mode fault detection.

1 = SS port pin includes MODF feature in Master mode
0 = SS port pin is not used by the SPI in Master mode

FUNCTIONS:

- The MODFEN (Mode Fault Enable) bit enables or disables mode fault detection in the SPI module. Setting MODFEN to 1 allows the SPI module to detect mode faults and generate interrupts if there is a mode configuration issue between master and slave. Setting MODFEN to 0 disables mode fault detection.

➢ The BIDIROE bit controls the output enable for the MOSI and MISO pins in bidirectional mode. It enables the output buffer for the MOSI pin in master mode and the MISO pin in slave mode when set to 1.
1 = Output buffer is enabled
0 = Output buffer is disabled

FUNCTIONS:

- The BIDIROE (Bidirectional Output Enable) bit enables or disables the output buffers on the MOSI and MISO pins when using bidirectional SPI communication. Setting BIDIROE to 1 enables the output drivers for the MOSI pin in master mode and the MISO pin in slave mode. This allows sending and receiving data. Setting BIDIROE to 0 disables the output buffers which isolates the pins.

➢ SPISWAI (SPI Stop in Wait Mode) bit is employed for power conservation during wait mode.

The SPIWAITE bit controls whether the SPI clock stops in wait mode. Setting SPIWAITE to 1 will halt the SPI clock when the device enters wait mode. Setting it to 0 allows the SPI clock to continue operating in wait mode.

FUNCTIONS:

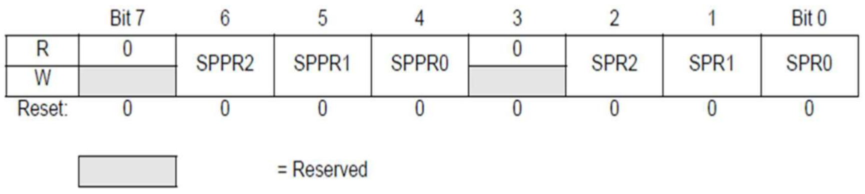
- The SPISWAI (SPI Stop in Wait Mode) bit controls whether the SPI clock stops when the microcontroller enters wait mode. Setting SPISWAI to 1 will halt the SPI clock generation when in wait mode. This reduces power consumption. Setting SPISWAI to 0 allows the SPI clock to continue running in wait mode so that SPI communications are uninterrupted.

➢ SPC0 — Serial Pin Control Bit 0

FUNCTIONS:

The SPC0 (SPI Pin Control 0) bit configures the SPI for either unidirectional or bidirectional communication. Setting SPC0 to 1 enables bidirectional mode, allowing simultaneous data transmission and reception through the MOSI and MISO pins. Setting SPC0 to 0 enables unidirectional mode where MOSI is output and MISO is input for half-duplex communication**.**

3) SPI Baud Rate Register [8 bit]:



	Bit 7	6	5	4	3	2	1	Bit 0
R	0	SPPR2	SPPR1	SPPR0	0	SPR2	SPR1	SPR0
W	▨				▨			
Reset:	0	0	0	0	0	0	0	0

▨ = Reserved

The SPPR2–SPPR0 (SPI Baud Rate Preselection) and SPR2–SPR0 (SPI Baud Rate Selection) bits determine the baud rates for the SPI communication, as illustrated in the table below. In master mode, modifying these bits will interrupt a transmission in progress, compelling the SPI system to transition into the idle state. In summary, these bits are responsible for setting the baud rate for SPI communication, and in master mode, any changes to these bits will terminate an ongoing transmission and put the SPI system into an idle state.

Functions of the SPPR2–SPPR0 bits:

- The SPPR bits (SPPR2, SPPR1, SPPR0) configure the baud rate prescaler which sets the SPI communication speed. Writing to these bits calculates the baud rate by dividing the peripheral clock frequency by the prescaler value. Changing the SPPR bits aborts any SPI transaction in progress and resets the communication.

4) SPI Status Register [8 bit] [Only Read & Not Write]:

	Bit 7	6	5	4	3	2	1	Bit 0
R	SPIF	0	SPTEF	MODF	0	0	0	0
W	▨	▨	▨	▨	▨	▨	▨	▨
Reset:	0	0	1	0	0	0	0	0

▨ = Reserved

- ➢ The SPIF (SPI Interrupt Flag) bit is raised once a received data byte has been successfully transferred into the SPI Data Register. To clear this flag, one must read the SPISR register with SPIF set, followed by a subsequent read access to the SPI Data Register.

  When SPIF is set to 1, it indicates that new data has been copied to SPIDR.
  When SPIF is set to 0, it signifies that the transfer is not yet complete

  Functions of the SPIF bit:

- The SPIF (SPI Interrupt Flag) bit signals when data has been received in SPI. Setting SPIF to 1 allows the SPI module to generate an interrupt when a data reception is complete, indicating new data is available to read. Setting SPIF to 0 disables receive interrupts. The SPIF flag needs to be cleared after each receive event.

- ➢ The SPTEF bit indicates when the SPI transmit data register is empty. Reading SPTEF can check if the transmit register is empty before writing new data to it. SPTEF needs to be cleared by reading the status register after transmitting, which resets the interrupt flag before loading new transmit data..

  When SPTEF is set to 1, it indicates that the SPI Data register is empty.
  When SPTEF is set to 0, it indicates that the SPI Data register is not empty.

  FUNCTIONS:

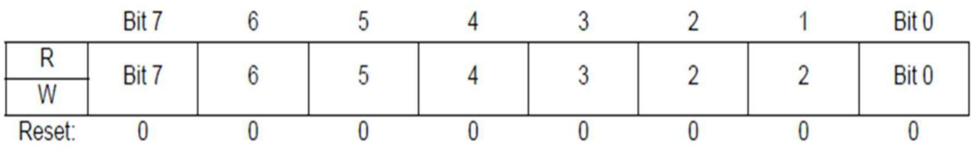
- The SPTEF (SPI Transmit Empty Flag) bit indicates if the transmit data buffer is empty in the SPI module. Reading SPTEF can check if the buffer is empty before writing new transmit data. SPTEF can also be used to trigger interrupts when the buffer becomes empty to signal readiness to accept more data. The flag needs clearing after reading and before writing new outbound data.

➢ The MODF bit flags a mode fault condition when SS goes low while in master mode if mode fault detection is enabled via MODFEN. It is cleared by reading the status register with MODF set then writing to the control register.
When MODF is set to 1, it indicates that a mode fault has occurred.
When MODF is set to 0, it signifies that no mode fault has occurred

FUNCTIONS:

- The MODF (Mode Fault Flag) bit indicates if a mode fault condition occurred in SPI communication. When set, it flags to the software that there is an SPI mode configuration mismatch between master and slave devices. MODF needs to be cleared by reading the status register and writing to the control register 1 to resume normal operation. Mode fault detection must be enabled by setting the MODFEN bit to use MODF.

5) SPI Data Register [8 bit]:



	Bit 7	6	5	4	3	2	1	Bit 0
R / W	Bit 7	6	5	4	3	2	2	Bit 0
Reset:	0	0	0	0	0	0	0	0

The SPI Data Register serves a dual role as both the input and output register for SPI data. Writing to this register enables the queuing and transmission of a data byte. In the case of a SPI configured as a master, a queued data byte is transmitted immediately after the completion of the previous transmission.

Here are the key points regarding the behaviour of the SPI Data Register:

Transmission: When a byte is written to the SPI Data Register, it is queued and transmitted promptly if the SPI is configured as a master.

Received Data Validity: The data present in the SPIDR becomes valid when the SPIF (SPI Transfer Complete Flag) is set.

Clearing SPIF and Receiving Data: If SPIF is cleared and a byte has been received, the received byte is transferred from the receive shift register to the SPIDR, and SPIF is set.

Unserviced SPIF and Subsequent Reception: If SPIF is set and not serviced, and a second byte has been received, the second received byte remains valid in the receive shift register until the initiation of another transmission.

Chapter 4: Results and Discussions on SPI Protocol Verification

This chapter elucidates the comprehensive verification of the SPI (Serial Peripheral Interface) protocol through simulated results and practical testing on an FPGA platform. It intricately details the outcome of RTL (Register-Transfer Level) coding written in Verilog HDL (Hardware Description Language), observed via the Vivado simulator, alongside hardware implementation results entailing testing the SPI peripheral with an Arduino as a slave and integrating an air quality monitoring system.

4.1 Evaluation of Results

The successful RTL coding for the SPI protocol demonstrates the feasibility and robust operation of the interface within a SoC (System on Chip) environment. Utilizing the Vivado simulator, waveforms for various operations, including write sequences, were meticulously examined.

4.1.1 Insight into the SoC Top Module

An overview of the SoC reveals a complex assembly hosting multiple protocols such as SPI, I2C, UART, and GPIO. Within this assemblage, SPI components are accentuated, underscoring their pivotal role in the system's communication framework as illustrated in Figure 4.1.

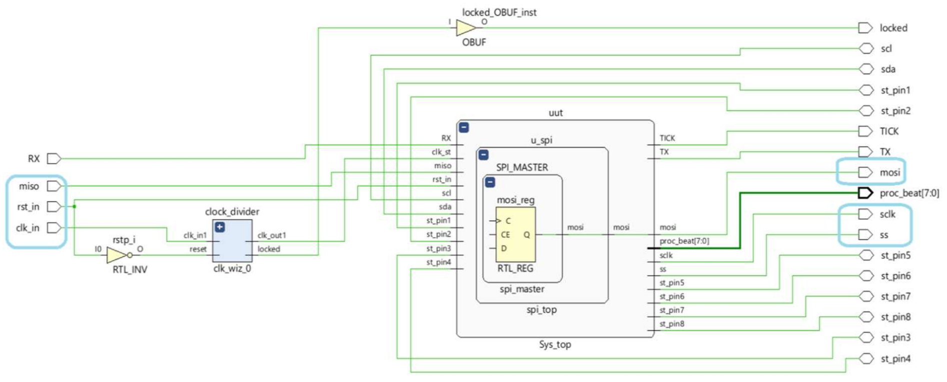
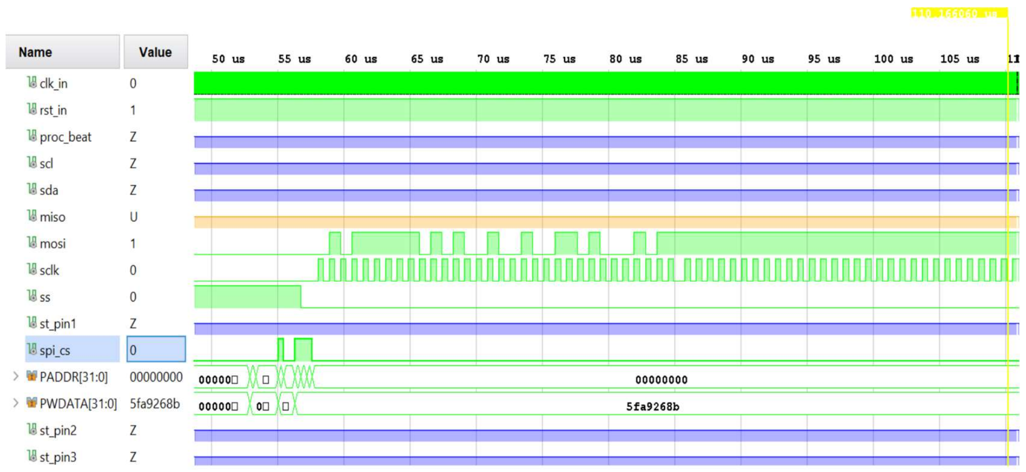


Figure 4.1: SoC Top Module Block

4.2 Simulation Outcomes and Analytical Overview

A precise analysis of the SPI Write operation underscores the efficient transfer mechanism spearheaded by the SPI protocol. As delineated in Figure 4.2, data (PWDATA) extracted from the Processor is adeptly conveyed through the MOSI (Master Out Slave In) line in synchronization with the sclk (serial clock), emphasizing the MSB-first data transfer policy.



Name	Value
clk_in	0
rst_in	1
proc_beat	Z
scl	Z
sda	Z
miso	U
mosi	1
sclk	0
ss	0
st_pin1	Z
spi_cs	0
PADDR[31:0]	00000000
PWDATA[31:0]	5fa9268b
st_pin2	Z
st_pin3	Z

4.3 Verification via Hardware Implementation

The real-world efficacy of the SPI peripheral was ascertained through two distinct experimental setups: interfacing with an Arduino Uno as a slave device and the deployment of an air quality monitoring system leveraging the SPI peripheral.

4.3.1 SPI Module and Arduino Uno Interaction

The configuration involving the Arty A7 board as the master and the Arduino Uno as the slave is presented in Figure 4.3. This setup required connecting the SPI module pins to the corresponding pins on the Arduino, following which, programming was carried out to enable data reception on the Arduino, substantiated by the 32-bit data display on the serial monitor as shown in Figure 4.4. This scenario exemplifies the bidirectional capability and reliability of the SPI module in real-world applications.

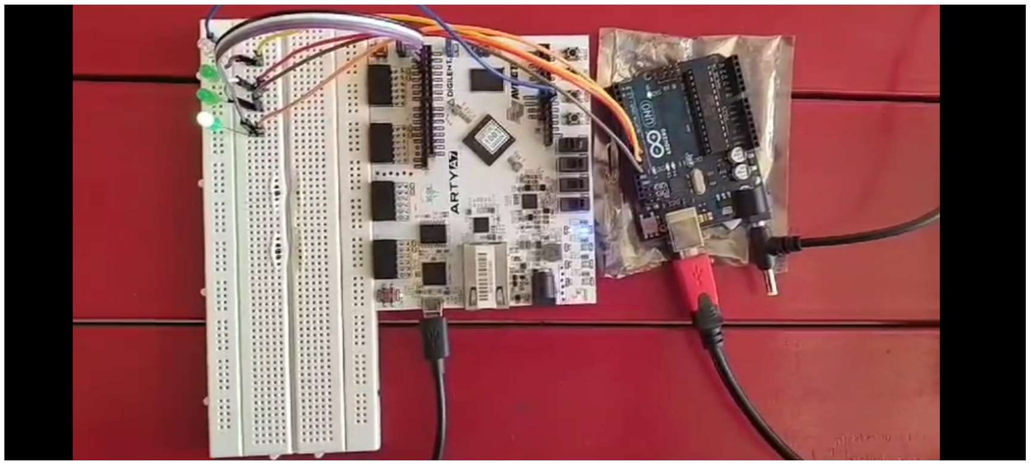
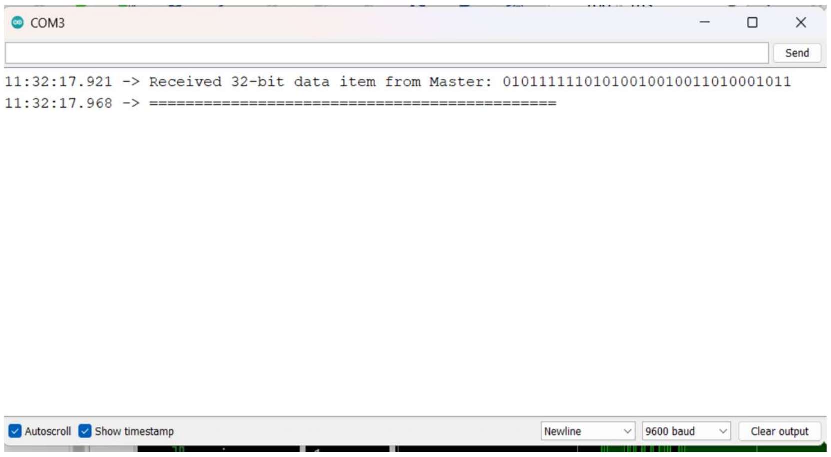


Figure 4.3: Hardware Implementation on ARTY7 FPGA Board with Arduino as a Slave



```
11:32:17.921 -> Received 32-bit data item from Master: 01011111101010010010011010001011
11:32:17.968 -> ============================================
```

Figure 4.4: Received 32-bit data from SPI master on serial monitor

Chapter 5

Conclusions and future scope

This chapter provides a synthesis of the project's achievements and outlines the path forward, highlighting the project's closure, key accomplishments, and the envisioned future developments in the domain of Serial Peripheral Interface (SPI) within System on Chip (SoC) architectures.

5.1 Conclusion

This project embarked on designing and implementing an interface between the processor and peripheral devices, with a keen emphasis on the Serial Peripheral Interface (SPI). The venture was twofold: to dissect the operational dynamics of SPI and to delve into the design intricacies associated with its implementation. By employing Verilog HDL for the SPI's implementation and the Universal Verification Methodology (UVM) for its verification, we mapped out the foundational elements of SPI—its operational principles, the configuration of its registers, and the initiation and management of serial communication between the master and selected slave devices. The project successfully delineated a comprehensive process for setting up an efficient SPI communication environment, alongside providing a detailed overview of the UVM-based verification platform for rigorously testing the SPI Design Under Test (DUT).

5.2 Future scope

Looking ahead, the project opens several avenues for broader application and deeper exploration within the field of RISC-V based SoC Design. The adaptability and efficiency of the SPI protocol evident from this project underscore its potential for broader applications, ranging from simple device control to complex data transfer systems within diverse SoC configurations. The project's future directions include.

Bibliography

[1] N Anand, George Joseph, Suwin Sam Oommen, and R Dhanabal. Design and implementation of a high-speed serial peripheral interface. In 2014 International Conference on Advances in Electrical Engineering (ICAEE), pages 1–3. IEEE, 2014.

[2] Izhar Izzudin bin Jamaludin and Hasliza binti Hassan. Design and analysis of serial peripheral interface for automotive controller. In 2020 IEEE Student Conference on Research and Development (SCOReD), pages 498–501. IEEE, 2020.

[3] S. Choudhury. Design and verification serial peripheral interface (SPI) protocol for low power applications. 2014.

[4] SPI Block Guide. Spi block guide v03. 06. S12SPIV3. pdf, 2003.

[5] Muhammad Hafeez and Azilah Saparon. Ip core of serial peripheral interface (spi) with amba apb interface. In 2019 IEEE 9th Symposium on Computer Applications & Industrial Electronics (ISCAIE), pages 55–59. IEEE, 2019.

[6] Fareha Naqvi. Design and implementation of serial peripheral interface protocol using Verilog hdl. International Journal of Engineering Development and Research, 2015.

[7] Veda Patil, Vijay Dahake, Dharmesh Verma, and Elton Pinto. Implementation of spi protocol in fpga Veda Patil. Editorial Board, page 142.

[8] T Durga Prasad and B Ramesh Babu. Design and simulation of spi master/slave using Verilog hdl. Int. J. Sci. Res, 3(8), 2014.

Report

ORIGINALITY REPORT

14%	11%	5%	%
SIMILARITY INDEX	INTERNET SOURCES	PUBLICATIONS	STUDENT PAPERS

PRIMARY SOURCES

1	www.nxp.com Internet Source	1%
2	users.ece.utexas.edu Internet Source	1%
3	www.slideshare.net Internet Source	1%
4	1library.net Internet Source	1%
5	es.scribd.com Internet Source	1%
6	citeseerx.ist.psu.edu Internet Source	1%
7	www.datasheet.hk Internet Source	1%
8	www.datasheetbase.com Internet Source	1%
9	datasheet.lcsc.com Internet Source	1%

10	1library.co Internet Source	1 %
11	technodocbox.com Internet Source	1 %
12	www.researchgate.net Internet Source	1 %
13	Izhar Izzudin bin Jamaludin, Hasliza binti Hassan. "Design and Analysis of Serial Peripheral Interface for Automotive Controller", 2020 IEEE Student Conference on Research and Development (SCOReD), 2020 Publication	1 %
14	docplayer.net Internet Source	<1 %
15	www.mcukorea.co.kr Internet Source	<1 %
16	z3d9b7u8.stackpathcdn.com Internet Source	<1 %
17	www.st.com Internet Source	<1 %
18	archive.org Internet Source	<1 %
19	docslide.us Internet Source	<1 %

20	Norman Dunbar. "Chapter 1 Arduino Interrupts", Springer Science and Business Media LLC, 2024 Publication	<1 %
21	cs.anu.edu.au Internet Source	<1 %
22	ece.unh.edu Internet Source	<1 %
23	C K Shaila, G Manoj, P.S. Divya, M Vijila.. "Functional Verification of SPI Protocol using UVM based on AMBA Architecture for Flash Memory Applications", 2023 4th International Conference on Signal Processing and Communication (ICSPC), 2023 Publication	<1 %
24	docs.rs-online.com Internet Source	<1 %
25	www.semanticscholar.org Internet Source	<1 %
26	datasheet.datasheetarchive.com Internet Source	<1 %
27	www.ijert.org Internet Source	<1 %
28	Saket Srivastava, Peter Hobden. "Low Cost FPGA Implementation of a SPI over High Speed Optical SerDes", 2018 IEEE	<1 %

	International Symposium on Smart Electronic Systems (iSES) (Formerly iNiS), 2018 Publication	
29	www.grin.com Internet Source	<1 %
30	hdl.handle.net Internet Source	<1 %
31	Muhammad Hafeez, Azilah Saparon. "IP Core of Serial Peripheral Interface (SPI) with AMBA APB Interface", 2019 IEEE 9th Symposium on Computer Applications & Industrial Electronics (ISCAIE), 2019 Publication	<1 %
32	docshare.tips Internet Source	<1 %
33	www.ijaerd.com Internet Source	<1 %
34	archives.njit.edu Internet Source	<1 %
35	core.ac.uk Internet Source	<1 %
36	www.coursehero.com Internet Source	<1 %
37	www.ijvdcs.org Internet Source	<1 %

38	www.sciencegate.app Internet Source	<1 %
39	Padmaprabha Jain, Satheesh Rao. "Design and Verification of Advanced Microcontroller Bus Architecture-Advanced Peripheral Bus (AMBA-APB) Protocol", 2021 Third International Conference on Intelligent Communication Technologies and Virtual Mobile Networks (ICICV), 2021 Publication	<1 %
40	dokumen.pub Internet Source	<1 %
41	kipdf.com Internet Source	<1 %
42	www.freescale.com Internet Source	<1 %
43	Qing Zhao, Xudong Chen, Jianguo Zhang, Pengfei Li, Junning Zhang. "Chapter 3 A Method ofSimulating SPI Interface Based onGeneral GPIO", Springer Science and Business Media LLC, 2022 Publication	<1 %
44	Hubert Henry Ward. "Programming Arduino Projects with the PIC Microcontroller", Springer Science and Business Media LLC, 2022 Publication	<1 %