Design and Implementation of SPI on FPGA

AET 006

Humum Shaikh
Electronics and Telecommunication
Vidyalankar Institute of Technology
Mumbai, India
humum.shaikh@vit.edu.in

Shraddha Jadhav
Electronics and Telecommunication
Vidyalankar Institute of Technology
Mumbai, India
shraddha.jadhav21@vit.edu.in

Abstract— ***The Serial Peripheral Interface (SPI) bus is a widely used communication protocol, primarily designed for short-distance, high-speed communication between microcontrollers and peripherals. This project focuses on the design and implementation of an SPI bus on a Field Programmable Gate Array (FPGA), offering a hardware-level solution that allows full-duplex communication between master and slave devices. The configuration of the SPI protocol is explored with respect to the clock phase and polarity control, as well as data transfer speeds and choice of control signals for a streamlined communication. The other option is to use the reconfigurability and parallel processing capabilities available on the FPGA and achieve both lower latency and higher customization potential compared to traditional microcontroller based implementations. Increasing the SPI communication, we add parity bits for error detection and acknowledge system confirming the successful data transmission. Simulation and hardware test are carried to validate and ensure compliance with SPI protocol. By doing so, a flexible and efficient SPI communication system is achieved which can be used in many embedded system applications with the balance of speed, accuracy and resource optimization.***

Keywords—FPGA, Verilog, SPI, Parity, Acknowledgement

I. INTRODUCTION

Motorola created SPI in the 1980s. This protocol continues being essential for embedded system communications. The protocol enables fast dual-channel data transmission that makes it ideal for linking microcontrollers to peripherals including sensors and memory devices while displaying information. The industrial sector as well as consumer electronics fields utilize SPI frequently because of its straightforward design and efficient functionality[1]. A main drawback of standard SPI protocol exists in its inability to perform automated error detection along with acknowledgment systems[3]. The lack of error detection methods during data transmission in practical systems affects data integrity seriously. There exists no method for validation or error detection of transmitted data which makes it impossible to confirm correct receipt. Such conditions cause system malfunctions primarily because of electrical disturbances and unreliable network connections. FPGAs present an optimal technology for SPI protocol implementation through their ability to reprogram easily and perform parallel operations alongside field programmability[2]. Programmers can make their own modifications and features to standard SPI implementations based on microcontrollers through this interface. The project uses FPGA's adaptable characteristics for SPI protocol improvement through an error-checking system implementation that adds parity bits to the system alongside receiving device acknowledgments. We intend to construct an SPI custom protocol through Verilog on Field-programmable gate arrays that enhances embedded system data reliability without compromising practicality or efficiency.

ABBREVIATIONS

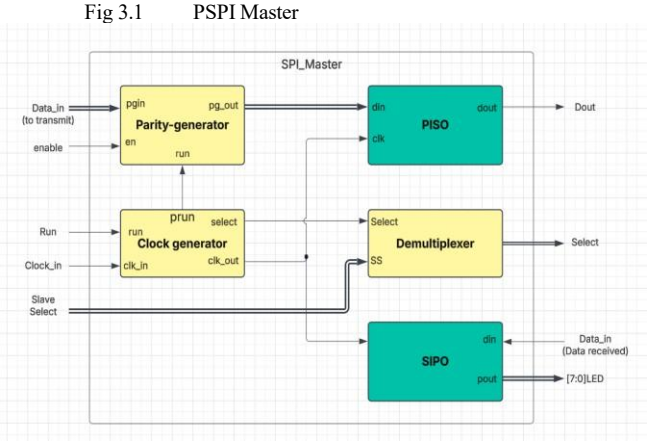
CE	Chip Enable
CS	Chip Select
FPGA	Field Programmable Gate Arrays
MISO	Master In Slave Out
MOSI	Master Out Slave In
PISO	Parallel In Serial Out
PSPI	SPI with Parity Generation
REG	Register
SCK	Serial Clock
SIPO	Serial In Parallel Out
SPI	Serial peripheral Interface

II. RELATED WORK

Fast data transmission occurs through the Serial Peripheral Interface (SPI) because this protocol has received extensive recognition for its capabilities when linking microcontrollers with peripheral devices. Extensive research concerning SPI design methodologies and optimization techniques and new applications proceeds due to FPGAs being popular in implementations because of their parallel processing and flexibility capabilities. According to Oudjida et al. (2009) laboratory results SP stands superior to I2C for protocol use because SP requires fewer FPGA resources and handles data transmission rapidly. Saha et al. (2014) designed an FPGA-based system that integrated built-in self-test capabilities to advance SPI communication reliability and establish better fault detection alongside performance stability features. SPI's relevance spans multiple industries. This technology supports fast sensor data transfer in real time for automotive ADAS systems as well as accelerating communication between IoT microcontrollers and peripherals and allows medical diagnostic testing using portable devices along with implantable monitoring devices. When FPGAs are used to implement SPI technology designers gain benefits and face challenges in verification alongside increased system complexity. The newly developed methods should implement simplified verification systems that enable QSPI support for faster data transmission. The literature demonstrates an expanding requirement for FPGA-based SPI optimization since it supports essential digital communication systems that require rapid operations in modern times.

III. System DEsign

Fig 3.1 PSPI Master



The system is designed to implement the PSPI protocol on an FPGA using Verilog. It includes two modules: master and slave, which work in sync during communication. The master begins by selecting the slave through an active-low slave select (SS) line. Once selected, both master and slave operate simultaneously. The master transmits 7-bit data along with a parity bit as the 8th bit. At the same time, the slave receives these 8 bits and calculates the parity of the first 7 bits. It then compares the calculated parity with the received 8th bit. If the parity matches, the data is considered correct; otherwise, it is assumed to be corrupted.

Simultaneously, the slave sends back an 8-bit response to the master. In the 9th clock cycle, it sends an acknowledgment bit. If the master receives an acknowledgment of 0, the transmission is considered successful. If the acknowledgment is 1, it indicates an error, and the master retries the transmission, allowing up to three attempts in total. Data is transmitted on the positive edge of the clock and received on the negative edge to ensure proper synchronization between both devices.

On hardware, buttons on the FPGA are used to send data, while LEDs are used to display the received data. This setup makes the protocol easy to test and observe and also helps in identifying transmission errors during development.

IV. Implementation

The standard SPI protocol lacks automatic error detection and correction functionality during its operation. The normal SPI transmission operates with an 8-bit data frame dimension. The data transfer of our enhanced version PSPI combines 7 data bits with 1 parity bit to maintain the standard frame size of 8-bits. The slave technology transmits standard 8-bit frames in its response. The transfer of information along with the master-to-slave acknowledgment requires 9 clock cycles though only 8 of those cycles dedicate to data movement. Successful transmissions occur when the master correctly receives no error signal through the acknowledgment (when the acknowledgment indicates no error). The master attempts three repeated attempts before receiving the right data from the slave. The communication begins exclusively from the master device since the slave lacks the ability to open transmission.

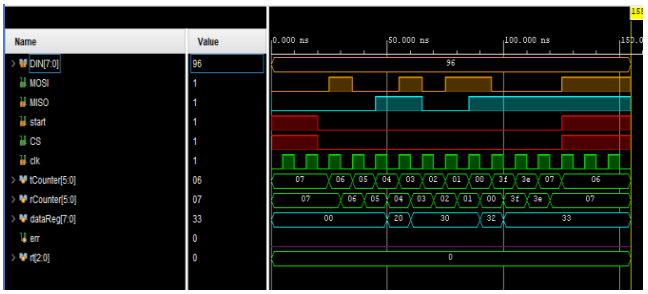
The timing of data transmission through the positive clock edge combined with negative edge reception creates a better data flow synchronization between master and slave devices. The slave select (SS) line must go low to trigger the slave operation in which it operates according to standard SPI functionality.

We employed FPGA board buttons for transmitting data while LEDs displayed the received information in hardware implementation. Visual verification of the protocol with this approach and easier debugging processes became possible. Our design started with an 8-bit data frame SPI communication before moving onto error detection followed by error correction implementation. We conducted tests using an ESP32 device as the slave component of our system. Slow operating speed of the ESP32 limited the data transfer rate with the FPGA platform. The implemented design on the FPGA reached its fastest operation at 12.5 MHz.

V. Result and Discussion

A. *Master Side Communication*

Fig 5.1 Master and Slave without error



The PSPI transaction starts with the chip select (CS) wire going low shown in red colored signals on the simulation window. Once this pin goes low, the module which is designed starts accepting the clock and it's rising and falling edges. On the positive edge of every clock cycle, the master sends the one bit of data through the MOSI pin shown in color orange in the simulation window.

Being full duplex, the module samples the incoming serial data through the Master In Slave Out (MISO) pin shown in the color cyan on the simulation window.

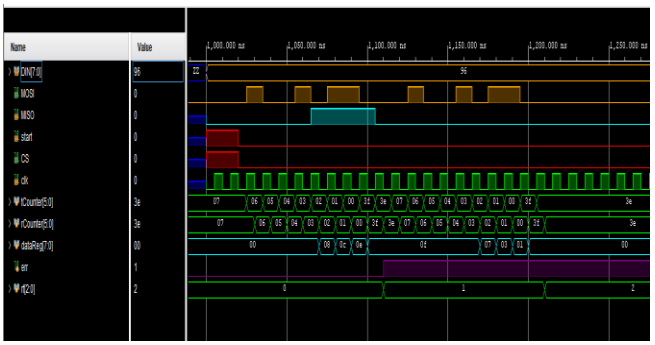
This incoming data is sampled on the falling edge of every clock cycle. All this data that is received from the slave is stored in an 8 bit internal register named as 'dataReg'.

The 9th bit that comes on the MISO pin is not a data bit but the acknowledgement bit, that is the 9th bit received is 1 which means that the slave did not find any issue with the parity of the data received from the master.

Hence the 'err' internal register shown in color purple in the master side remains 0 so as to indicate that there was successful reception of data by the slave and hence there will be no need for retransmission of the data bits.

Registers 'tCounter' and 'rCounter' are both counters to store the count of the data bits transmitted and are used internally to keep track of the next bit to be transmitted and the location where the next bit will be stored in the internal register 'dataReg'.

Fig 5.2 Master to Slave with error and retransmission



Here in the above simulation results, there was purposefully error introduced to check and verify the functioning of the PSPI design. In this module, the data to be transmitted is again 0x96 i.e. 10010110, although this data meets the even parity requirement of the slave, the assumption is that the data flipped somewhere in the middle or wrongly received by the slave due to which the parity won't match.

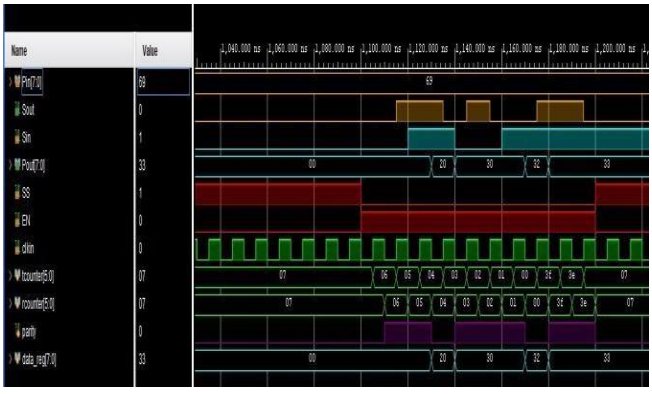
The data transmission and reception starts as the CS pin goes low, the module starts accepting the clock and it's rising and falling edges and works accordingly. Here as well, the data bits are transmitted on the positive edge of every clock cycle once the CS pin goes low, and the data is transmitted in the first 8 clock edges.

The data is also sampled at the falling edges just like previous operation. However here if observed carefully then when the data is sampled at the 9th falling edge, the MISO pin is low which means that the slave has given the acknowledgement that the parity does not meet in the data it received from the master.

Due to this, the master knows that now it has to retransmit the same data again. The 'err' reg which is a single bit internal register shown in color purple is having value 1 which means there is error.

B. *Slave Side Communication*

Fig 5.3 Slave to Master without error



Here in the slave side, the data to be transmitted from the slave to the master is shown in color orange in the simulation window. The data received to the slave from the master is shown in the color cyan which is serial data and this data is stored in the register called as 'Pout' which is an 8 bit register. Here also the slave select or the CS pin is shown in color red, the clock cycle is shown in color green and it's supporting counter registers which are used to keep track of the next bit to be transmitted and to know the location in the receiving register where the received bit from the master will be stored are also shown in green. The parity register is in purple indicating whether the parity is matched or unmatched.

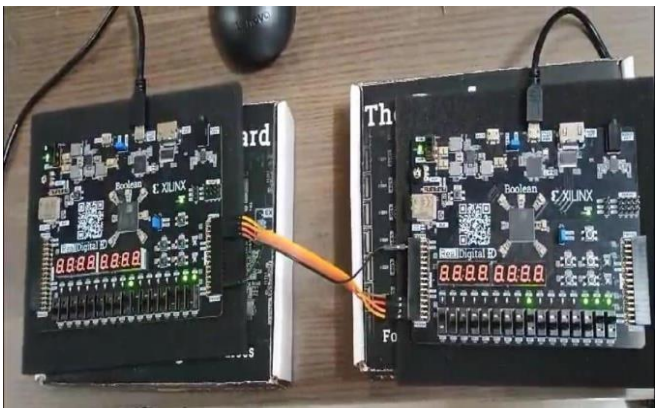
Once the chip select pin goes low, the slave module starts accepting the clock cycle and works as per the rising and falling edges of the clock cycle. The data is transmitted on the rising edge of every clock cycle, note that here all the 8 bits transmitted are data bits.

The data received from the master on the MOSI line is sampled on the falling edge of every clock cycle.

The parity is calculated as per the data bits in the 'Pout' register and depending on that, the 9th bit decides whether to give acknowledgement for error or no error. If it is 1 then the parity matches and no error detected and if it is 0 then the parity does not match and there is an error in the data. This bit is transmitted as the 9th bit and sent to the master over the MISO line. For getting the proper results, two tests were performed, in the first test, it was simulated such that there was no error in the transaction and in the second test, an error was introduced purposefully. The one where there was no error, there was acknowledgement given from the slave side about it so that there is no redundant retransmission of data. The one where there was an error introduced, the slave side sent the acknowledgement that there is an error and hence the master made retransmission happen.

C. *Hardware Implementation*

Fig 5.4 Real-Time communication between Master and slave



More changes were made to the existing master and slave modules to fit the working on FPGA boards.

Their constraint files were written and the bitstream was generated accordingly. The two FPGA boards shown above are Spartan 7 FPGA development boards. The one on the left is working as a Master and the one on the right is the Slave. They are connected as per the MOSI, MISO, CS, SCLK configuration of SPI through the PMOD connectors provided on the board and their resultant outputs are shown on the LEDs above the slide switches. The slide switches on both the sides are used to set the data to be transmitted for both the master and the slave. On the master side, this switches also set the slave to be selected and the enable pin. The implementation was successful and we were able to have a

communication at a maximum clock rate of 12.5 Mhz limiting to the internal hardware delay of the FPGA IC

VI. Conclusion and Future Work

In this project, we designed and implemented an improved version of the SPI protocol called PSPI. The goal was to add basic error detection and an acknowledgment mechanism to make communication more reliable. We kept the standard 8-bit frame size by sending 7 bits of data and 1 parity bit from the master. The slave sends back 8 bits, and a 9th clock cycle is used for acknowledgment. If no error is detected, the transmission ends. If there's an error, the master retries up to three times. This whole process happens only at the master side since the slave can't initiate communication.

In our setup, data is sent on the positive edge of the clock and received on the negative edge. The slave is selected using an active-low slave select line. We used buttons on the FPGA board to send data and LEDs to show the received data, which made testing easier. We first implemented basic SPI, then added error detection, and finally error correction. We also tested our design with an ESP32 as the slave. Since ESP32 runs slower than the FPGA, we had to keep the speed low. On FPGA, we were able to run the protocol at a maximum clock speed of 12.5 MHz.

Currently, PSPI works at lower clock speeds compared to standard SPI, which can go up to 25 MHz. This can be improved in future versions. Also, there’s still a chance that the parity bit or acknowledgment bit might get flipped, which means the protocol isn’t completely error-proof. To improve this, stronger error detection methods like CRC (Cyclic Redundancy Check) can be used.

Another limitation is that in standard SPI, each slave needs its own select line. Inspired by the I²C protocol, we can add addresses to slaves so that multiple devices can be handled using fewer lines. This would make the protocol more efficient and scalable.

Acknowledgment

We would like to express our gratitude to all those who supported and guided us throughout the successful completion of our project titled “Design and Implementation of SPI on FPGA.”We are especially thankful to our internal guide, Prof. Satendra Mane (Vidyalankar Institute of Technology, Mumbai) for his encouragement, valuable guidance, and support throughout the duration of the project. His suggestions played an important role in the successful completion of our work. We would also like to extend our sincere thanks to our external guide, Mr. Shubhum Kadam (CDAC, Pune) for providing us with valuable technical inputs and real-world perspectives that helped us in our approaches and increase our understanding of the topics. We are grateful to the Department of Electronics and Telecommunication at VIT for providing access to the Spartan 7 Boolean boards and other resources. Special thanks to the lab assistant Mr. Kiran Hemale for allowing us to sit and work in the lab during our development stages. Finally, we acknowledge the support of our classmates, friends, and families who stood by us and encouraged us throughout this journey.

References

[1] F. Leens, "An introduction to I2C and SPI protocols," IEEE Instrumentation & Measurement Magazine, vol. 12, no. 1, pp. 8–13, Feb. 2009.

[2] K. Oudjida et al., "FPGA implementation of I2C & SPI protocols: A comparative study," in Proc. 16th IEEE Int. Conf. Electronics, Circuits and Systems (ICECS), 2009, pp. 507–510.

[3] S. Saha, M. A. Rahman, and A. Thakur, "Design and Implementation of SPI Bus Protocol with Built-In-Self-Test Capability over FPGA," in Proc. 1st Int. Conf. Electrical Engineering and Information and Communication Technology (ICEEICT), 2014.

[4] Y. H. K. Laxmi, “Data Transfer of Flash Commands Through Different Serial Peripheral Interfaces (SPI) Approaches,” Int. J. Eng. Res. Technol. (IJERT), vol. 4, issue 22, 2016.