Benha Journal of Applied Sciences (BJAS) **Print: ISSN 2356–9751**
Vol. (9) Issue (5) (2024), (127-134) **Online: ISSN 2356–976x**

http://bjas.bu.edu.eg

A State Machine-Based Approach for Implementing SPI Communication on FPGAs

Ayman S. Shama[1], Manar H. Lashin[2], Ayman A. Nada[3]
[1]*Mechanical Engineering Department, Engineering Faculty, Benha University, Benha, Egypt*
[2]*Electrical Engineering Department, Engineering Faculty, Benha University, Benha, Egypt*
[3]*Mechatronics and Robotics Engineering Department, Egypt-Japan University of Science and Technology, Alexandria, Egypt*
Email: ayman.shamah@bhit.bu.edu.eg

Abstract

The Serial Peripheral Interface (SPI) is a widely used synchronous serial communication bus that facilitates communication between a master device, and one or more subordinate devices, referred to as slave devices. This article outlines the conceptualization and design procedures of an interface for the SPI communication bus on a Field Programmable Gate Array (FPGA) platform, using the LabVIEW programming environment alongside the LabVIEW FPGA module. The structure and operation of the SPI communication bus are presented, along with its timing structure and four distinct operational modes. The implementation of SPI bus communication on the FPGA is achieved via the use of the state machine methodology. With the state-machine SPI technique, multiple devices can be managed simultaneously through a single port. Additionally, the clock frequency can be adjusted online, making the system adaptable and flexible for different tasks. LabVIEW is used to implement the SPI module circuit, which is then mapped onto the National Instruments Single-Board (NI SbRIO-9631) FPGA board. The L3G4200D gyroscope sensor is used to conduct testing on the protocol and verify the effectiveness of the state machine technique. The results of this study demonstrate the successful implementation of the SPI communication bus interface on an FPGA using LabVIEW and LabVIEW FPGA module. The module exhibits versatility, efficiency, and the ability to communicate with a wide range of devices.

Keywords: Serial Peripheral Interface (SPI), LabVIEW, Serial Communication, FPGA, NI SbRIO-9631.

1. Introduction

The Serial Peripheral Interface (SPI) is a synchronous serial interface technology first introduced by Motorola. The SPI bus is physically implemented by a module on a microprocessor control unit (MCU) connected to a peripheral microcontroller (PIC micro), which allows the MCU to perform high-speed data communication with various peripheral devices in full-duplex synchronous serial mode [1]. In mechatronics applications, FPGAs are often used to control sensors, actuators and other devices that perform various tasks. To communicate with these devices, microcontrollers need to use protocols that define how data is exchanged [2,3]. SPI is one of many serial buses that has several advantages over other widely used buses, such as the I2C, CAN, and USB. For example, an SPI bus can transmit data at several Megabytes per second (Mbps) faster than other serial buses. This paper discusses the structure, operation, timing structure and four operational modes of the SPI communication bus [4].

In addition to the SPI conceptualization and design procedures outlined in this paper, the utilization of the SPI offers several notable advantages for communication within embedded systems. The SPI protocol provides a straightforward and efficient means of data exchange between a master device and multiple slave devices, enabling seamless integration and communication across diverse hardware components.

One significant advantage of employing SPI is its inherent simplicity and flexibility, making it ideal for a wide range of applications in embedded systems. With its synchronous serial communication architecture, SPI facilitates high-speed data transfer while requiring minimal hardware resources, thereby optimizing system performance and resource utilization. Moreover, the implementation of SPI communication on a Field Programmable Gate Array (FPGA) platform, as demonstrated in this study, offers additional benefits. By leveraging the state machine methodology and the capabilities of LabVIEW alongside the LabVIEW FPGA module, developers can achieve robust and flexible SPI interfaces tailored to their unique application needs.

Overall, the successful implementation of the SPI communication bus interface on an FPGA using LabVIEW and LabVIEW FPGA module underscores the versatility, efficiency, and effectiveness of SPI in facilitating seamless communication and integration within embedded systems. These advantages make SPI a compelling choice for a wide range of applications, including sensor interfacing, communication with peripherals, and control systems design.

2. SPI Protocol Overview

The SPI bus is a four-wire synchronous serial bus commonly used to communicate between a master device and one or more slave devices [5]. The master device controls the communication by generating the clock signal (SCLK) and selecting the slave device to communicate with using the chip select (CS) signal. The SPI bus uses an 8-bit serial shift register in the master and slave devices. Data is transmitted on the MOSI (Master Out Slave In) line from the master to the slave, and data is received on the MISO

(Master In Slave Out) line from the slave to the master. The data is transferred one bit at a time and synchronized to the SCLK signal [6].

The SPI bus has four essential signal lines:

- CS (Chip Select): This line is used to select the slave device for communication. The signal is typically active low, so pulling it low will select the slave device.
- SCLK (Serial Clock): This line synchronizes the data transfer between the master and slave devices.
- MOSI (Master-Out-Slave-In): This line transmits data from the master to the slave.
- MISO (Master-In-Slave-Out): This line receives data from the slave to the master.

2.1. SPI Bus Working Principle

The SPI bus is in idle state when there is no data transferred between the master and the slave. The master initiates communication by enabling the chip select (CS) signal and sending the clock signal over the (SCLK) line. Data is simultaneously transmitted (shifted out serially from the master onto the MOSI line) and received (from the slave's MISO line) on the rising or falling edge of the clock signal, depending on the clock polarity (CPOL) and clock phase (CPHA) settings [7]. The data transfer process is completed when the master returns the SCLK line to the idle level and the CS line to the inactive state. The SPI master/salve interface is illustrated in figure 1.

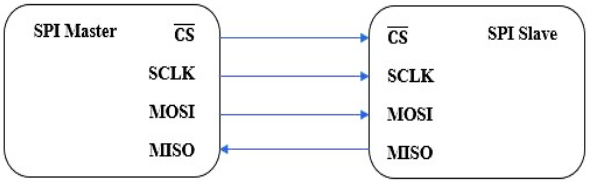

Fig. (1) SPI bus structure with master and slave.

The CS line carries the signal that initiates the communication of the corresponding slave. The SCLK line carries the clock signal that synchronizes the data transfer between the master and the slave. The MOSI line carries the master output signal with the data to be transmitted to the slave. The MISO line carries the slave output signal with the data received from the slave. The CPOL and CPHA settings determine when the data is shifted from the master and when the slave samples it. The CPOL setting specifies the idle state of the SCLK line, while the CPHA setting specifies when the data is shifted out from the master relative to the rising or falling edge of the SCLK line [8].

The SPI bus working principle is summarized as follows:

1) The host sets the SCLK output to an idle level, the CS line is pulled high to an inactive state, and the SPI bus is idle when there is no data to be transferred between the master and the slave.
2) The master initiates communication by pulling the CS line low.
3) The master then sends the clock signal (SCLK) line.
4) The data is shifted out serially from the master onto the MOSI line and received on the slave's MISO line on the rising or falling edge of the SCLK line, depending on the CPOL and CPHA settings.
5) The data transfer process is completed when the master sends the last bit of data and returns the SCLK line to the idle level.
6) The master then pulls the CS line high to de-select the slave.

2.2. SPI Clock Specifications

The SPI clock signal has two parameters: clock polarity (CPOL) and clock phase (CPHA) as shown in figure 2. The CPOL parameter controls the level of the clock signal idle state. In contrast, the CPHA parameter controls whether data is sampled on the falling or rising edge of the serial clock. The following table shows the four possible combinations of CPOL and CPHA [9,10].

Table (1) Four Combinations of CPOL and CPHA

SPI Mode	CPOL	CPHA	CPOL in Idle State	Sample and/or shift data
0	0	0	Logic Low	Data sampled on the rising edge and shifted out on the falling edge
1	0	1	Logic Low	Data sampled on the falling edge and shifted out on the rising edge
2	1	0	Logic High	Data sampled on the falling edge and shifted out on the rising edge
3	1	1	Logic High	Data sampled on the rising edge and shifted out on the falling edge

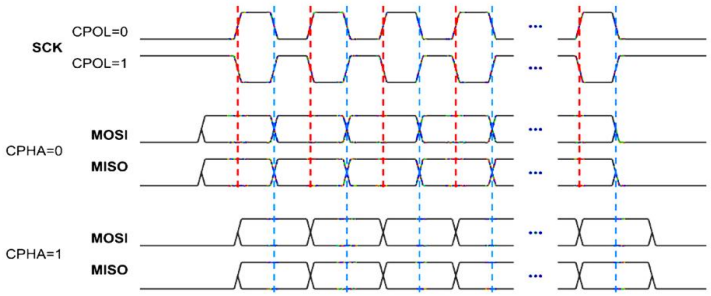

Fig (2) SPI timing diagram

3. Software Design of SPI Bus Communication

The software design of SPI communication bus interface is implemented using LabVIEW and LabVIEW FPGA modules.

The main tasks of the SPI master module are to:

- Receive the necessary commands from the host to execute the relevant code blocks.
- Generate the clock signal based on the CPOL and CPHA parameters to ensure synchronization.
- Set the CS line logic low to start transmission and high to stop it.
- Receive 8-bit parallel data from the host, convert it to serial data, and then send the bits one a time to the slave over the MOSI line.
- Obtain parallel data from the slave, convert it to serial data, and output it through the MISO line one bit a time.

The SPI module consists of three main Virtual Instruments (VIs): the Host Application Program Interface (API), the FPGA Multiplexer VI, and the SPI Engine VI. The Host Interface is responsible for communication between the host PC or real-time controller and FPGA. The FPGA Multiplexer is responsible for routing the data between the host and SPI Engine. The SPI Engine is responsible for applying the SPI protocol sequence to the hardware. Both the FPGA Multiplexer and the SPI Engine are implemented on FPGA using two parallel loops to take advantage of the parallel processing capabilities of the FPGA. This allows the SPI module to achieve high data transfer rates [11,12].

3.1. Host Application Program Interface

The host API is a LabVIEW interface to the SPI communication that simplifies data transfer between a host PC or real-time controller and the FPGA multiplexer, figure 4 shows the operating sequence of this VI. This API consists of two VIs:

- The (SPI_Configure) establishes the port to be used, along with the SCLK rate, CPOL, CPHA, and CS. The Configure command passes this data to the FPGA multiplexer and stores it in the global variable of the FPGA.

The(SPI_Write/Read) takes the number of bits transferred to/from the specified port. The Write/Read command passes the written/read data to/from the FPGA multiplexer target-scoped FIFO.

3.2. Field-Programmable Gate Array Multiplexer

This Virtual Instrument consists of a state machine that manages the handshaking between the host and the FPGA and multiplexes data to the correct port. The host is responsible for determining the state that the multiplexer should execute. The multiplexer initially sits in idle state, waiting for a start command from the host. When the start command is received, the multiplexer then checks the command sent from the host. If the command is "Configure", the CPOL, CPHA, and CS values are written to global variables to prepare for the following data transfer. If the command is "Write/Read SPI", data received from the host is passed to a target-scoped FIFO, then transferred to the VI-scoped FIFO to be passed to the SPI engine in the Write and Read states.

3.3. SPI Engine

A state machine is an excellent choice for implementing the SPI protocol because the protocol has a specific order in which bus transitions occur [13]. Each state completes one part of the communication before moving on to the next step in the proper order, figure 5 shows the operating sequence of this VI.

The following are the steps involved in the SPI state machine:

1) The state machine initiates when receiving a start command from the FPGA multiplexer, writing the configuration parameters to be passed to corresponding states.
2) The state machine reads a byte of data from a VI-scoped FIFO, passed from the multiplexer and stored as a Boolean array on the FPGA through MOSI line.
3) The chip select (CS) line is pulled low to prepare the slave device to receive and send data one bit at a time.
4) The clock signal is generated depending on the clock mode. If CPOL is 0, the clock idle state is logic low and the first edge of clock signal is a rising edge, If CPOL is 1, the clock idle state is logic high and the first edge of the clock signal is a falling edge.
5) If CPHA is 0, the data is clocked in when the clock signal changes from idle to active state. If CPHA is 1, the data is clocked in when the clock signal changes from active to idle state.
6) A waiting state is inserted between setting and resetting the clock signal to guarantee the proper clock rate. This process is repeated for each bit that needs to be sent
7) After each byte transfer, the data received from the slave device is stored in another VI-scoped FIFO before being sent back to the host via the FPGA multiplexer through MISO line.

If all the data sent to the state machine has been sent to MOSI line and the corresponding data has been received from MISO line, the CS line is reset to the idle state, and the program waits for the following command.

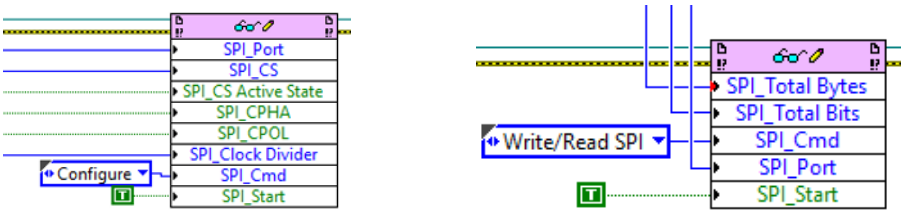

Fig. (3) SPI_Configure Parameters & SPI_Write/Read Parameter

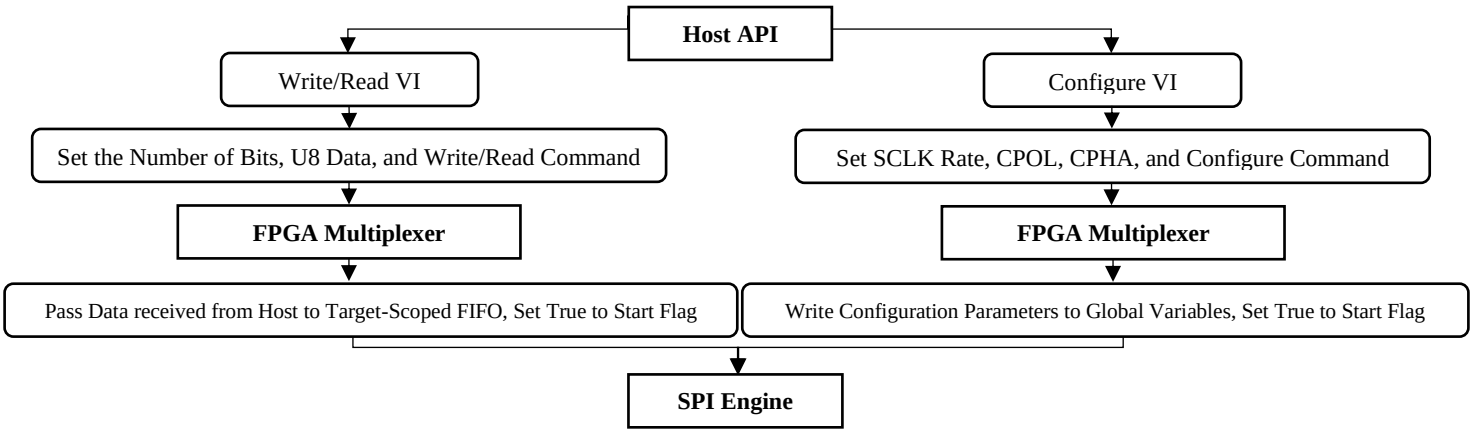

Fig. (4) SPI host API sequence

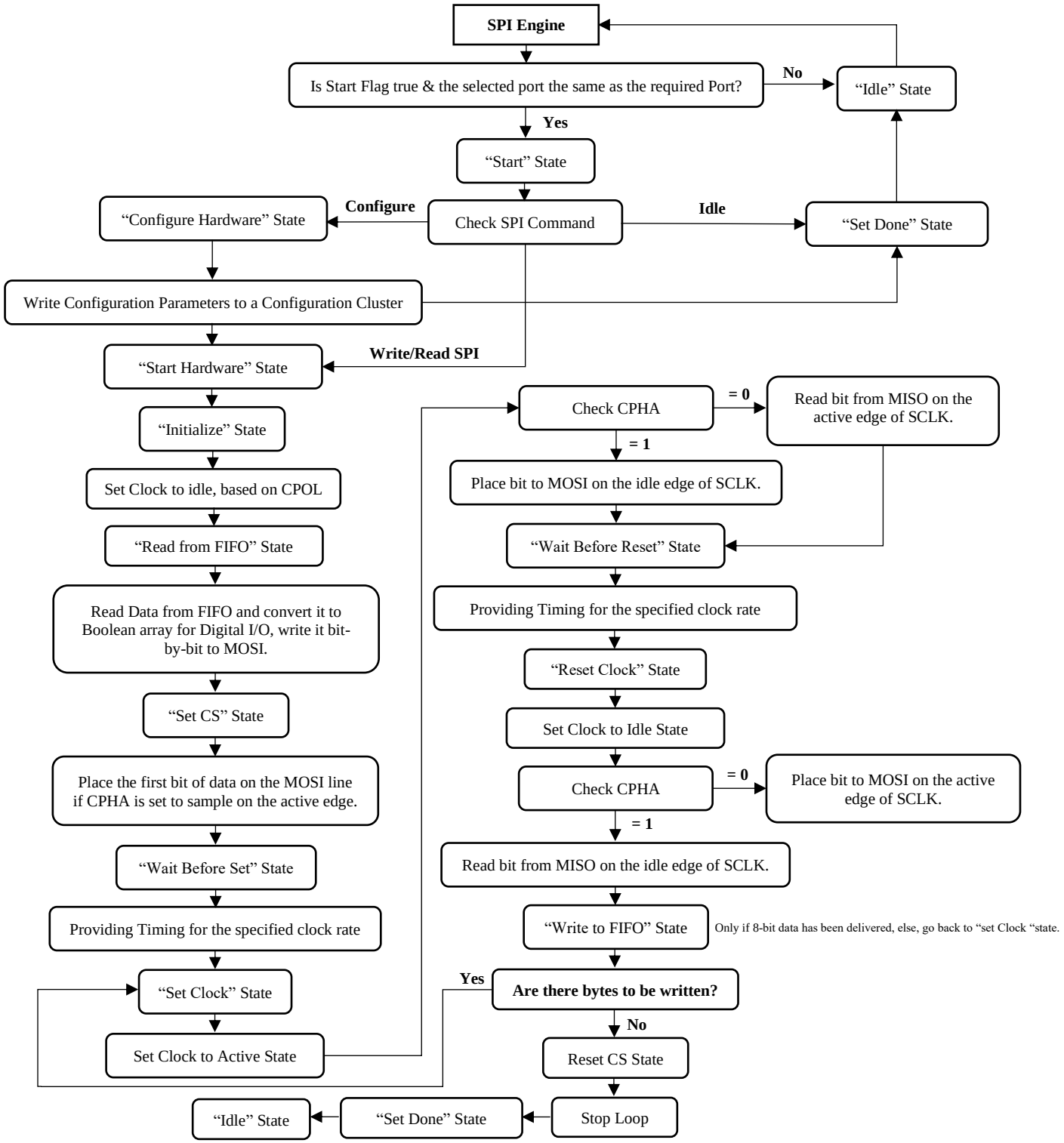

Fig. (5) SPI Engine state machine operating sequence

4. System Development

In the proposed implementation of SPI on FPGA, it has been obtained to use National Instrument Single Board Reconfigurable Input/Output (SbRIO-9631) interfacing with the L3G4200D gyroscope sensor to validate the implemented protocol.

4.1. Hardware Interfacing

4.1.1. NI SbRIO-9631

The development environment for the NI SbRIO-9631 prototype robot is intricate and modularly constructed.

Figure 6 shows a simplified block diagram of the robot's electro-mechanical components and all its fundamental modules**.** The entire hardware setup is built around a main board with an FPGA processor from Xilinx, connected to peripheral circuits and digital and analog input/output channels. [14].

4.1.2. L3G4200D Gyroscope Sensor

The STMicroelectronics L3G4200D, is a three-axis angular rate sensor known as PmodGYRO which can deliver unprecedented zero rate level stability and sensitivity over temperature and time. It has a sensing component and an IC interface that can communicate the measured angular rate to the external world using an I2C or SPI digital interface [15].

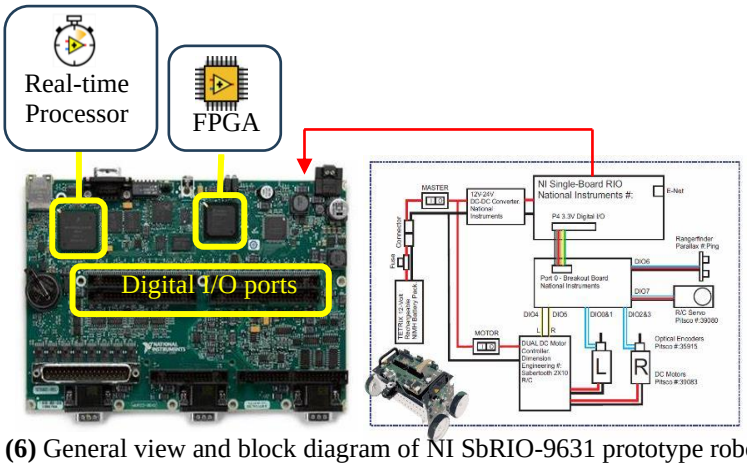

Fig. (6) General view and block diagram of NI SbRIO-9631 prototype robot

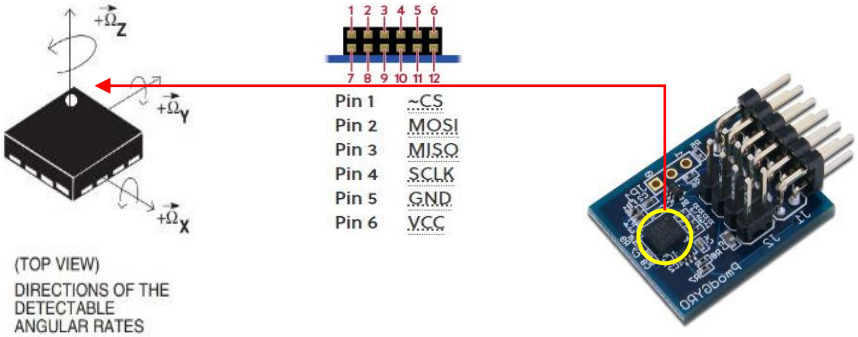

Fig. (7) PmodGYRO pin configuration

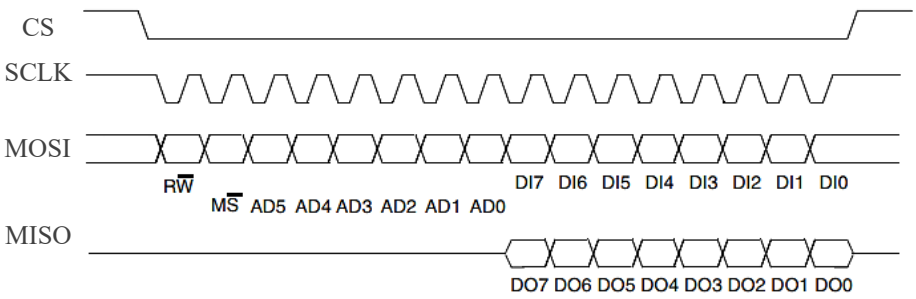

Fig. (8) PmodGYRO (L3G4200D) timing diagram

The following features are described with reference to the L3G4200D datasheet:

- The SPI protocol is used by the sensor to connect to the host board. SPI mode is activated by driving the Chip Select (CS) line to a logic low. Figure 8 shows how to communicate with the sensor to read a specific register's data.
- The SCLK's idle level is high, making CPOL = 1, and the serial port's data input and output are driven at the SCLK's falling edge and are intended to be captured at its rising edge, meaning CPHA = 1. Therefore, mode 3 must be asserted.

- Referring to MOSI line:

1) Bit 0: RW bit
   a- When 0, the data DI (7:0) is written to the device.
   b- When 1, the data DO (7:0) is read from the device, which drives MISO at the start of bit 8.
2) Bit 1: MS bit.
   a- When 0, the address remains unchanged in multiple write/read commands.
   b- When 1, the address is auto incremented in multiple write/read commands.
3) Bit 2-7: address AD (5:0). This is the address field of the indexed register.
4) Bit 8-15: data DI (7:0) (write mode). This is the data that is written to the device.
5) Bit 8-15: data DO (7:0) (read mode). This is the data that is read from the device.

- The data output X, Y and Z are 16 bit-rate values placed on registers addresses from (28h) to (2Dh), each axis has two 8-bit registers.
- Some specific registers must be defined first such as registers responsible for selecting the output data rate, resolution, power mode, …etc., please refer to the sensor datasheet for more information.

4.2. Software Interfacing

4.2.1. SPI Implementation on FPGA

Using the Xilinx compilation tool, the LabVIEW FPGA code is implemented on the SbRIO-9631 FPGA board after that the design flow was successfully emulated and verified in a practical project. This tool constructs the software code as a hardware circuit on the FPGA through configurable logic blocks (CLBs), I/O blocks and programmable interconnects that join various logic blocks to produce the functionality specified in the software. It also routes signals from I/O blocks into and out from the FPGA.

4.2.2. Host Implementation on Real-Time Target

Figure 9 shows the sequence in which the code is implemented and the corresponding block diagram.

5. Experimental Results and Test Operations

Both the simulation and the actual experiment demonstrate that the bus is in an idle state when the CS value is 1 and that the bus begins to transmit data when the CS value is 0. On the falling edge of the first cycle, the host sends the highest bit of data to the slave through MOSI, filling up the Target-scoped FIFO and raising the write flag high when transmitting to the 8th bit, indicating that the 8-bit data transfer is complete. The slave then corresponds to the data sent and sends back the data in the required register via the MISO line, filling the VI-scoped FIFO and being ready to be sent back to the host via the multiplexer. In conclusion, the SPI module's intended function is working properly.

6. Conclusion

The study focuses on presenting the SPI communication protocol implementation strategy using LabVIEW software with LabVIEW FPGA module to design the SPI module that complies with the SPI bus specification, followed by functional testing to verify the implemented protocol. The protocol is designed to require minimum resources on the FPGA board, making it an excellent choice for use in projects. In addition, the SPI module performs as intended, operates reliably, and scales effectively. Moreover, this state-machine SPI implementation technique enables the control of various slaves on the same port and facilitates online changes for dynamic clock frequencies.

Because of FPGA reconfigurability, as well as the SPI bus's wide variety of application, the module can be simply extended with hardware as required, to boost functionality and capabilities.

Table (2) FPGA Utilization Summary of the SPI

Device Utilization	Used	Available	Percent
Slice Registers	2124	15360	13.8%
Slice LUTs	3336	15360	21.7%
Block RAMs	4	24	16.7%
Mult18X18s	1	24	4.2%

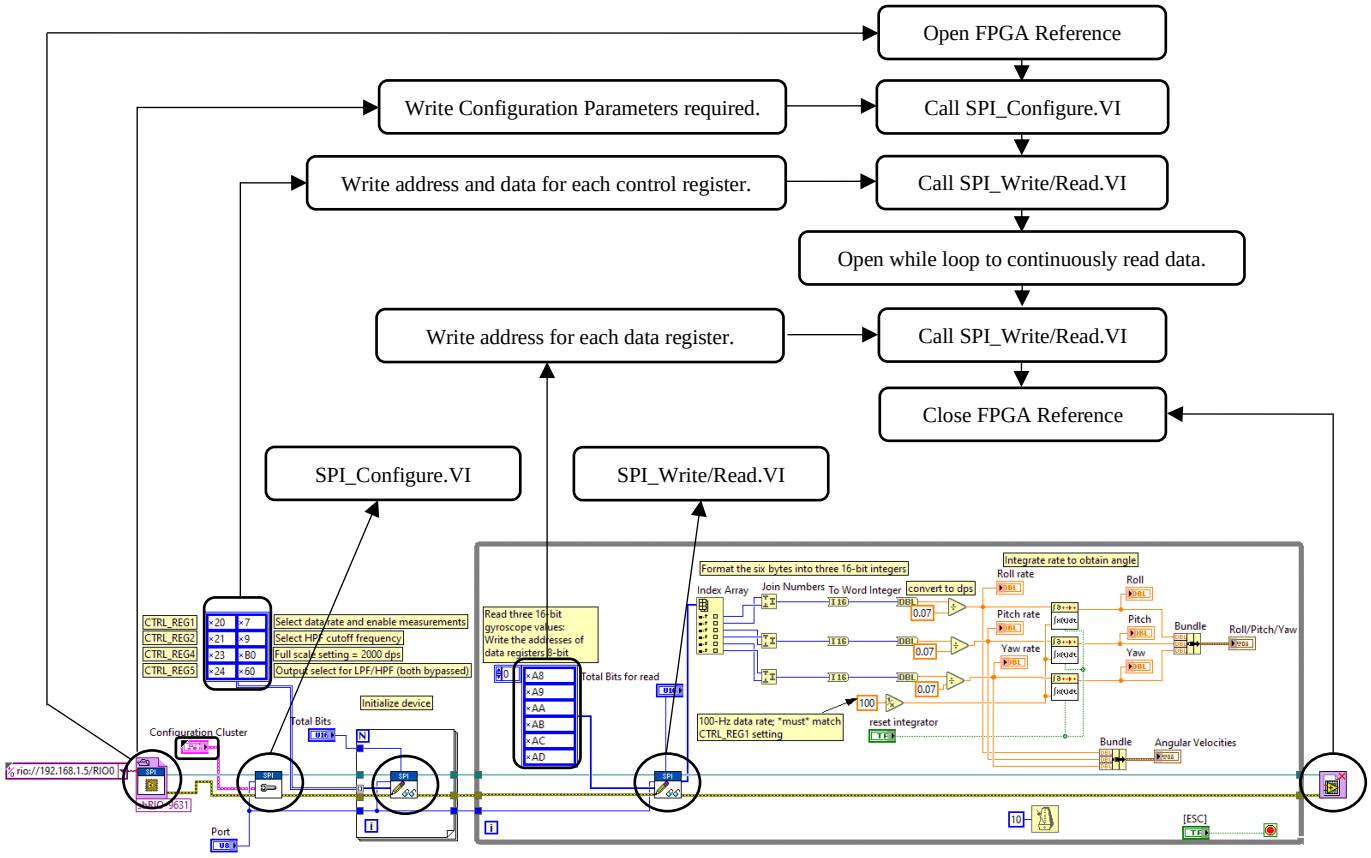

Fig (9) Host program block diagram.

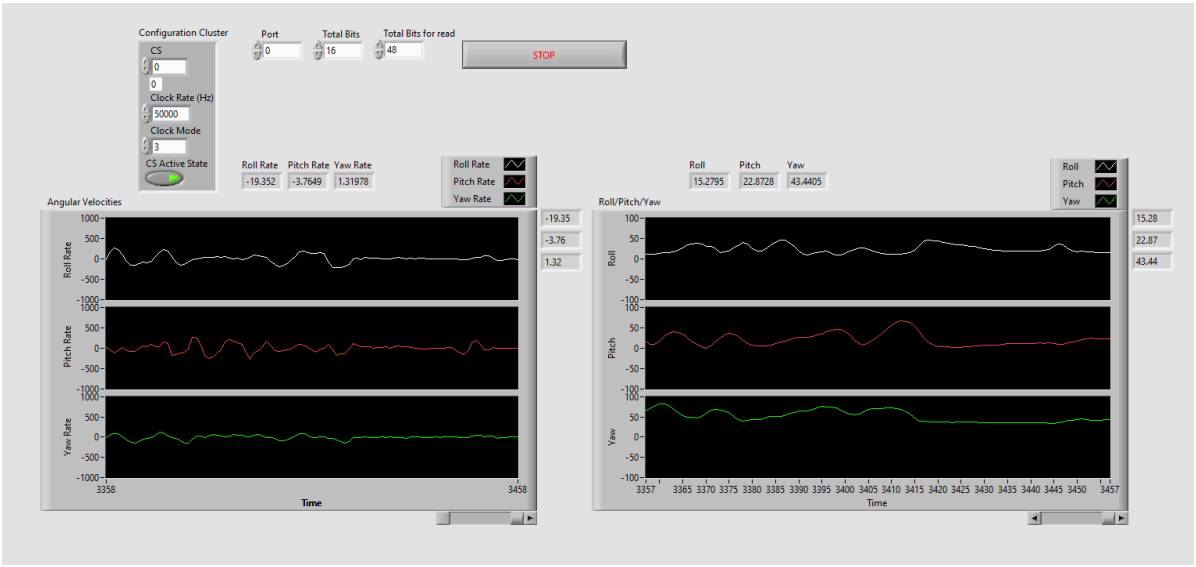

Fig (10) Front panel showing X, Y and Z angular velocities & angular displacement.

References

[1] F. Leens, (2009) An Introduction to I2C and SPI Protocols.In: IEEE Instrumentation & Measurement Magazine. Beijing. pp. 8-13

[2] Ayman A. Nada, Victor Parque, Mona A. Bayoumi, 2023, "Accelerating the Performance of Fuzzy-FPGA Based Control in LabVIEW for Trajectory Tracking Problems", 22nd IFAC World Congress, Yokohama, Japan, July 9-14, 2023.

[3] A.K. Oudjida, M.L. Berrandjia, A. Liacha, R. Tiar, K. Tahraoui & Y.N. Alhoumays, " Design and Test of General-Purpose SPI Master/Slave Ips on OPB Bus," 2010 IEEE.

[4] AA Nada, AH Bashiri, 2018, "Integration of multibody system dynamics with sliding mode control using FPGA technique for trajectory tracking problems", ASME 2018 Dynamic Systems and Control Conference, September 30–October 3, 2018, Atlanta, Georgia, USA. Paper No: DSCC2018-9108.

[5] Lattice Semiconductor Inc., "SPI Slave Controller V 01.0," October 2012.

[6] W.Mai, W.Liu. (2007) Design and Implementation of SPI Interface Based on FPGA and MSP430. Instrumentation Users, pp. 100-102.

[7] Motorola Inc., "SPI Block Guide V03.06," March 2003.

[8] W.C.Zhu, S. ,Zhang, H.L.Jiang. (2017) Design of high speed data communication interface based on ARM and FPGA. Journal of Guilin University of Electronic Technology, pp. 293-297

[9] Oudjida.A.Berrandjia, M.Tiar.R, Liacha.A. & Tahraoui, K., "FPGA Implementation of I2C & SPI Protocols:a Comparative Study," IEEE, 2018

[10] D.N. Oruganti, Design of power efficient SPI interface, in 2014 IEEE International Conference on Advanced Communications, Control and Computing Technologies (IEEE, 2014)

[11] Jianlong Zhang, Jiwei Wang, Chunyu Wu, Wenjing Zhang, "The design and realization of a comprehensive SPI interface controller," Second International Conference on Mechanic Automation and Control Engineering (MACE), 2011 IEEE.

[12] N.Q. B. M. Noor and A. Saparon, "FPGA implementation of high speed serial peripheral interface for motion controller," in Proc. 2012 IEEE Symposium on Industrial Electronics and Applications (ISIEA), pp.78-83, Sept. 2012.

[13] N.Q. B. M. Noor and A. Saparon, "FPGA implementation of high speed serial peripheral interface for motion controller," in Proc. 2012 IEEE Symposium on Industrial Electronics and Applications (ISIEA), pp.78- 83, Sept. 2012.

[14] User Manual National Inst. Co. available: https://www.ni.com/en-lb/support/model.sbrio-9631.html

[15] User Manual Digilent Co. available: https://digilent.com/reference/pmod/pmodgyro/start