2021 6th International Electrical Engineering Conference (IEEC 2021)
April, 2021 at NEDUET, Karachi, Pakistan

FPGA Implementation of a High-Speed SPI Design For Single Board Computers (SBCs)

Farooq Alam[1] and Fariha Farooq[2]

[1] Department of Electronics and Power Engineering,
National University of Sciences and Technology, Islamabad, Pakistan (farooq.phdee19pnec@pnec.nust.edu.pk)
[2] College of Engineering, PAF-KIET, Karachi, Pakistan

Abstract: This paper proposes a Field Programmable Gate Arrays (FPGA) based implementation of a high-speed Serial Peripheral Interface (SPI) protocol for readout connectivity with the Single Board Computer (SBC). Micro-Electromechanical Systems (MEMs) have customized readout circuits and an SPI interface can help standardize their connectivity with off-chip SBCs. We develop SPI interface for the standard Commercial-Off-The-Shelf (COTS) accelerometer and gyroscope available from Analog/Digital Interface (ADI) or STMicroelectronics (STM). We employed the Xilinx FPGA board to interface with accelerometer and gyroscope sensors which regularly collects high-speed data for further high-level processes. Furthermore, we proposed a parallel interface for both sensors operating simultaneously exploiting FPGA hardware parallelism which renders the overall speed of data communication significantly increased. Subsequently, we want to connect the MEMs chip to the SBCs. Therefore, we used a Raspberry pi board which facilitate us for testing and implementation of the proposed methodology. The proposed SPI protocol has an operating frequency 10MHz and it is adaptable to different device frequencies. Simulation, as well as hardware results, are provided.

Keywords: SPI, Xilinx FPGA, Verilog HDL, MEM SBC integrations, Raspberry pi.

I. INTRODUCTION

SPI interface is often considered one of the simplest and best communication protocols among the Peripheral Component Interconnect (PCI)-Express, Universal Serial Bus (USB) and others in the world of digital communication systems. SPI can connect with several numbers of communication devices and facilitate a fast-speed bidirectional data communication between both devices. On the contrary, other serial data communication protocols such that I2C and Universal Asynchronous Receiver/Transmitter (UART) require more time to bidirectional data transportation on a single wire. Furthermore, SPI communication between sensors is very common and easy to implement. Several works have been done to the sensor interface and data communication between SBCs and Complementary metal-oxide-semiconductor (CMOS) devices. ([1] - [3])

Many works have been proposed in the literature to optimize the speed of SPI communications. A power-down mode of SPI operation is proposed in [4]. Shift registers are used as double buffer registers to avoid overflow data loss. Apart from that, the technique using clock signal in this architecture has reduced 13% power utilization by employing shift registers. Design and testing of high-quality SPI interface for On-chip Peripheral Bus (OPB) are discussed in [5]. Full description proposed SPI Intellectual Property (IP) is implemented on in Verilog HDL at Xilinx Virtex 5 FPGA board. High-speed SPI communication for motion controller is implemented on FPGA [6].

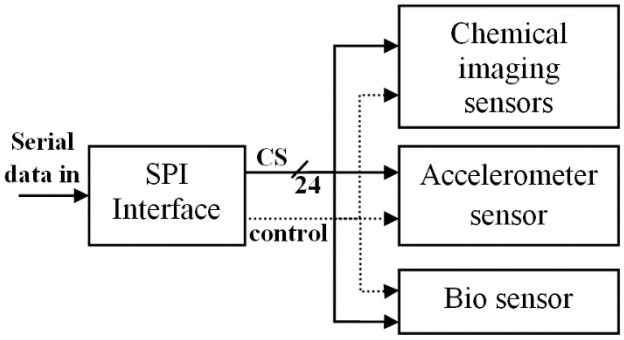


Fig. 1: Read-out Interface (ROI) Block Diagram

SPI is simulated using Mentor's ModelSIM and deployed on ALTERA DE2, Cyclone FPGA board. Experimental results are presented between different data transfer modes with correct data types renders continuous data transfer.

Fig. 1 shows the SPI connection with analog type sensors. The There are three different sensors connected to the SBCs directly through the SPI protocol. The SPI facilitates the sensors top collect the data synchronously by using Chip Select (CS) with the utility of FPGA clock frequency. Furthermore, the control bits are transmitted to each sensor.

With the advancement of technology, there is a need for fast and reliable serial communications protocols. Therefore, FPGAs can be utilized for data communications as well as fast data processing and control. FPGA Finite State Machine (FSM) architecture for UART protocol is developed for the Radio-Frequency Identification (RFID) tag data communication [7].

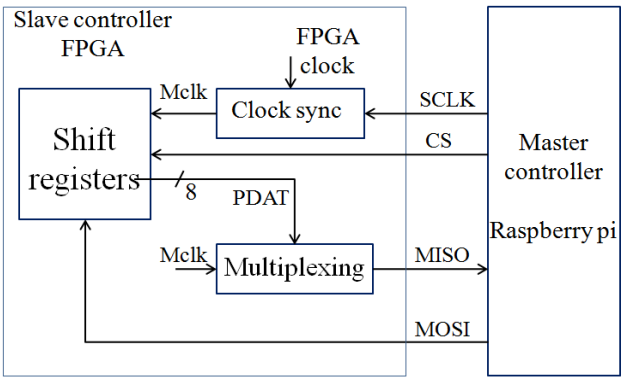


Fig. 2: Proposed SPI architecture

Optimized communication design is presented for vehicle parking systems.

Since the Raspberry operates Linux OS and the FPGA is designed at digital logic components, we choose to deploy the SPI design on the FPGA board, which does not require the use of a synchronous clock.

A shift register delivers by the SPI Master synchronizes the communication protocol, regardless of the built-in clock generation module of each side.

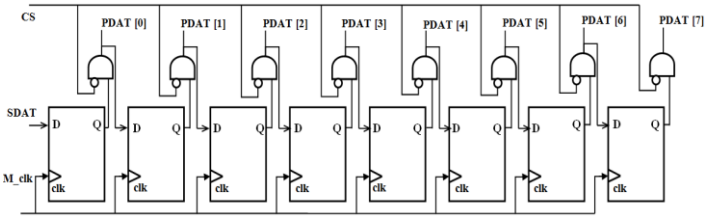
We picked Python software for the Linux OS (Raspberry) and Verilog HDL for the FPGA in terms of computer languages. As a consequence, on the Linux software, the SPI operator can be combined with widely used architectures. [8]

Raspberry PI 2 model B used as the SPI slave with FPGA Master. A General-Purpose Input/Output (GPIO) port is included in the Raspberry PI (GPIO). This GPIO's four lines were used to introduce SPI. The SPI Master library is implemented in Python. The application server is installed on the Raspberry Pi to allow for the development of a user-friendly application for experimenting and testing utilization. The Python programming and modules are used to build an SPI Master engine.

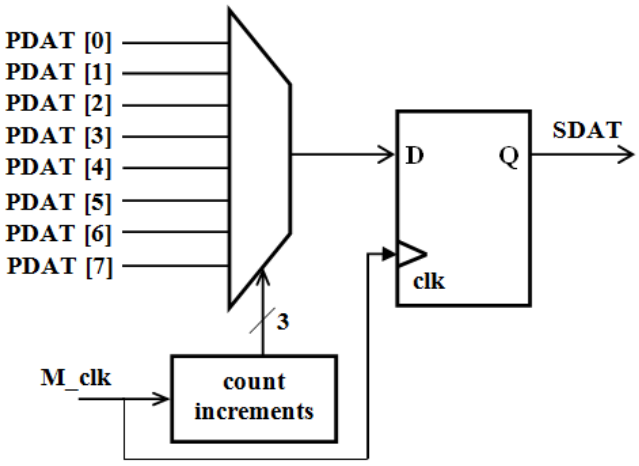
This paper is organized as follows. Section II elaborates design specifications of the proposed SPI interface. Section III presents the proposed strategy. Section IV explains the simulation results. Section V demonstrate practical results and section VI draws the final conclusions and point forward future work.

II. DESIGN SPECIFICATIONS OF SPI INTERFACES

The industry-standard SPI master controller is a 4-wire signal interface. Data width is 8-bit standard word length. However, nonstandard 3 to 16-bit word length communication can be performed. Standard SPI signal consists of Chip Select (CS), Serial Clock (SCLK), Master Out Slave In (MOSI) and Master In Slave Out (MISO). Fig. 2 shows the complete block diagram of our SPI interface.



(a) SPI shift registers design



(b) Parallel to serial data multiplexing design

Fig. 3: Slave controller component level design

A raspberry pi 3 model B is employed as a master controller which mainly produces SCLK and CS output signals. Raspberry pi 3 has built-in BCM2835 and WiringPi libraries for SPI interface. It works on Linux (Raspbian) which is an open-source operating system. Therefore, programming in Python was developed in a free programming builder Geeny editor. The real-time SPI data results in display on the Linux command window environment. The python programming performs following functions. [9]

- Generate CS signal periodically to receive the SPI data.
- Generate SCLK signal for the SPI slave input.
- Generate MOSI data to deliver at the FPGA input.
- Receive MISO input for the Raspberry pi module.

On another hand, Xilinx Spartan 6, LX9 FPGA micro-board is used as a slave which receives command signals from the Master controller. FPGA is a fast processing device that works for data generation and returns it to the master controller. Idea is to develop a fast and efficient SPI interface that facilitates wide data range transfer between SBC and the external world. Therefore, the embedded controller for the SPI interface requires both the high data rate transmission as well as the processing speed. Xilinx Spartan 6, micro-board has a 100 MHz on-board CMOS CDCE913 modular Phase

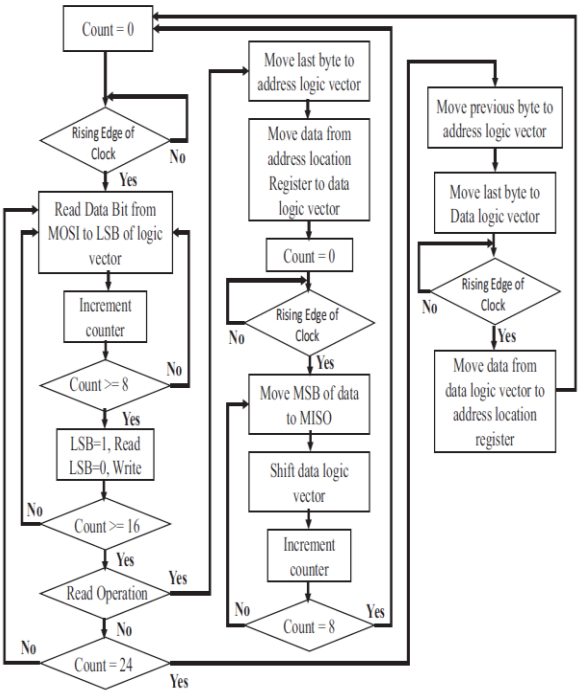


Fig. 4. Flow chart of FPGA for SPI bus interfacing

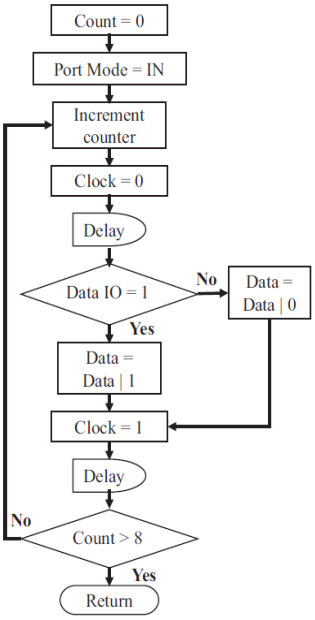


Fig. 5. Raspberry pi read interfacing flow algorithm.

Lock Loop (PLL) based low-cost and high-performance programmable clock. Hence, this technology facilitates achieving more optimized and fast SPI communications. [10]

III. PROPOSED STRATEGY

The proposed design component level design of SPI interface and implemented on FPGA. The design of an SPI is similar to shift registrars. Parallel in serial-out shift registers stores data to proceed and work on the correct side to further process. A simplified architecture

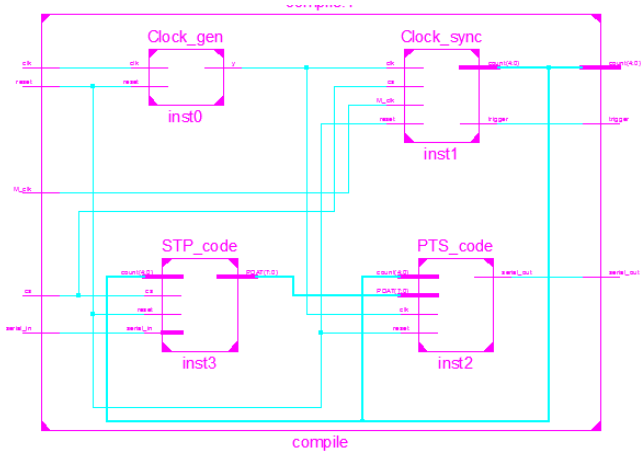


Fig. 6: Complete Register Transfer Level (RTL) view of shift registers shown in Fig. 3a.

Since MOSI serial data needs to be converted into parallel for verifications and data processing in slave controller, master clock synchronized sequential logics are used for shifting of Serial Data (SDAT). Furthermore, combinational logic gates render a simultaneous Parallel Data (PDAT) data delivery of all stages based on the CS logic conditions. Data storage and a number of stages cause transition delays. Therefore, our customizable serial to parallel conversion module can operate up to 128 SDAT to PDAT conversions. Fig. 3b shows the schematic of a parallel data PDAT to serial SDAT conversion. In this module, we multiplexing the 8-bit PDAT parallel input data to convert into SDAT serial output data. There exists a counter which generates 3-bit binary count increments fed into the channel selection of multiplexer. Consequently, sequential data generate an exhibit from PDAT Least Significant Bit (LSB) to Most Significant Bit (MSB). The generated data needs to be clock-driven with master control. Therefore, Flip Flop (FF) is used to provide master clock synchronized output. Fig. 4 shows the flow diagram of the FPGA software. As CS becomes small, FPGA checks bit by bit the rising edge of the Raspberry pi clock from MOSI pin. However, when the FPGA reads a segment and determines whether it is a read or writes functions depending on the byte flags. The address Byte is also interpreted by the FPGA. FPGA utilizes a logic function to temporarily store the address location for reading operations. FPGA transfers the message bit by bit from the registry at the MISO pin site. The FPGA also sends the next byte in the writing process. A temporary function logic is the value of FPGA's data. Besides this, FPGA writes data values in the registry.

Fig. 5 shows the flow map of the Raspberry pi flow for the serial interface. For write operations, the method takes bytes and bytes to write. As the Raspberry pi IO port mode is set to in or out for both transmitting and receiving respectively. This subsidiary writes or checks a

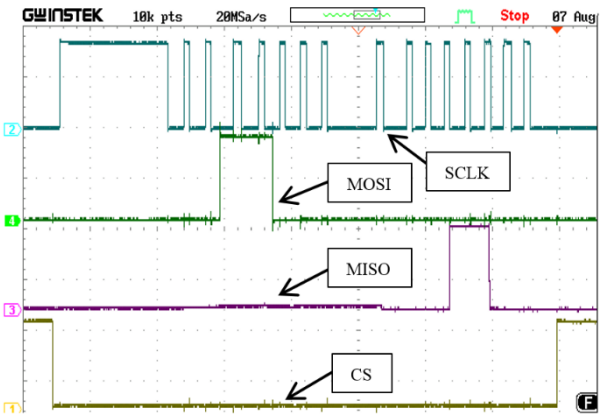


Fig. 7: Results on oscilloscope

full byte with the IO pin, bit by bit. The write flag will be placed in the standardized flow chart and the address will be preceded, which shows the next operation. Similarly, for reading, the Raspberry pi reads the data byte or writes data according to the flow of the algorithm.

IV. SIMULATION RESULTS

The top-level RTL view of the overall module is presented in Fig. 6. The Verilog HDL module has four inputs reset, SCLK, CS, MOSI and relevant single output port MISO which connects with the Raspberry pi. Here clock generator module divides 1 MHz SPI clock frequency which is further fed into a clock synchronizing module used to generate trigger signal and count increments while counting number of Master input clock cycles. The up counting of master clock cycle facilitate is further used to drive both shift register and multiplexer modules. To synchronize SPI interface with slave, counter counts total of 16 clock cycles before restarting again due to CS idle. Afterward, that count fed into serial to parallel data conversion module to convert SDAT MOSI input data into PDAT processing. In the end, PDAT needs to convert into serial data SDAT again to transfer it to the master controller. Eventually, the transmitted serial data is displayed on the command window screen of master controller.

Fig. 7 shows a time-line simulation of our SPI design. Initially CS signal is low. Since SPI communication remains idle while CS high. Therefore, SCLK, MOSI and MISO remain at default condition. When turning low, serial communication has started and SCLK begins to transfer from master to slave controller.

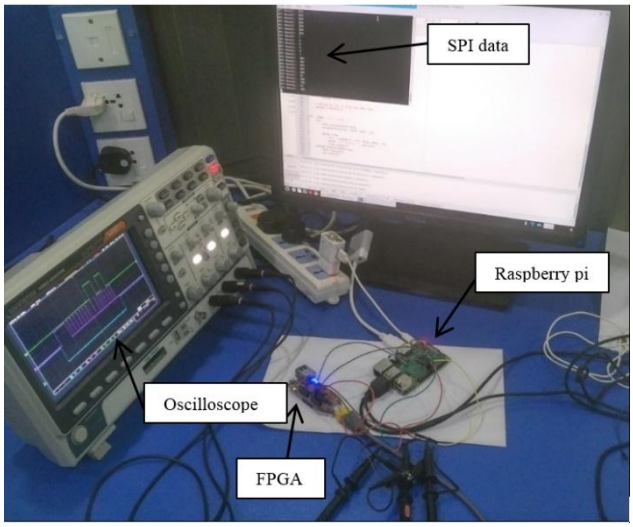


Fig. 8: Hardware setup

Subsequently, MOSI data transfer starts from master to slave controller. serial MOSI transferred data bits 8'b11100111 or 8'd224 data bits.

V. HARDWARE IMPLEMENTATION

To implement SPI serial communication between master and slave, we employed raspberry pi 3 model B as a master controller and Xilinx Spartan 6 LX9 Micro-Board as a slave physical controller. FPGA Peripheral Module (PMOD) connection facilitates to interface Micro-board with the external world. Raspberry pi has dedicated General Purpose Input/Output (GPIO) ports for SPI interface. Hence, we can link both embedded controllers so that we able to initialize serial communication between them. Fig. 8 shows the hardware test bench where serial data can be observed on the Raspberry pi command window as shown in Fig. 9. Moreover, data transmission between devices can be observed at the oscilloscope which results are presented in Fig7. The summary of resource utilization is presented in Table I.

TABLE I: FPGA resource utilization summery

Device type	Used	Available	%
Slice registers	15	11,440	0%
Slice LUTs	21	5,720	0%
Slice LUTs as logic	7	5,720	1%
Slice LUTs as flip flops	14	22	63%
Bounded IOBs	11	200	5%

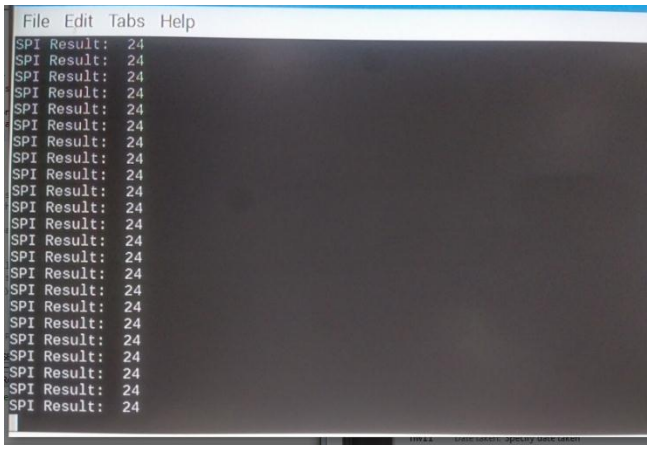


Fig. 9: Master controller SPI data display

VI. CONCLUSION

This paper presents an FPGA-based implementation of a high-speed SPI interface. We develop an SPI interface for the standard COTS accelerometer and gyroscope available from ADI or STM. this research. This relation is used to construct a high-level program. This exemplifies a workable approach. On the FPGA, Verilog HDL is used. On the side, and using a high-level user interface approach on the other hand. We've restricted the program to digital inputs/outputs. The job can be generalized to cover other areas of the organization. FPGA stands for Field Programmable Gate Array. This makes it possible to change the behavior. Dynamically of an FPGA It's also possible to finish the job in a shorter period. the idea of speeding up the communication, which is correlated to the speed of the Master's clock. We employed the Xilinx FPGA board to interface with accelerometer and gyroscope sensors which regularly collects high-speed data for further high-level processes.

REFERENCES

[1] Slawomir Michalak, “Raspberry Pi as a measurement system control unit”, International Conference on Signals and Electronic Systems (ICSES), 2014.

[2] Xuhui Chen, Hongyun Yang*, “Binary Image Acquisition Method of CMOS Image Sensor based on MCU SPI interface”, International Conference on Advanced Computing and Applications, 2015.

[3] Anand N ; George Joseph ; Suwin Sam Oommen ; R Dhanabal, “Design and implementation of a high speed Serial Peripheral Interface”, IEEE International Conference on Advances in Electrical Engineering (ICAEE), 2014.

[4] Dwaraka N Oruganti ; Siva S Yellampalli, “Design of power efficient SPI Interface”, International Conference on Advances in Computing, Communications and Informatics, 2015.

[5] 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”, 7th IEEE International Multi-Conference on Systems, Signals and Devices, 2010.

[6] NurQamarina binti Mohd Noor ; Azilah Saparon, “FPGA implementation of high speed serial peripheral interface for motion controller”, IEEE Symposium on Industrial Electronics and Applications, 2012.

[7] Wajeeha Umar ; Farooq Alam ; S.M. Usman Ali Shah, “Optimized design of A Parking Management System Using FPGA”, IEEE International Conference on Open Source Systems & Technologies, 2014.

[8] NurQamarina binti Mohd Noor, Azilah Saparon, “FPGA Implementation of High Speed Serial Peripheral Interface (SPI) for Motion Controller”, IEEE Symposium on Industrial Electronics and Application 2012.

[9] Haissam Hajjar, Hussein Mourad, “Implementation of an FPGA - Raspberry Pi SPI Connection”, International Conference on Advances in Circuits, Electronics and Micro-electronics, 2019.

[10] Shantanu Telharkar, Shreerang Dabade, Tejas Karangale, Shardul Telharkar, “FPGA Implementation of Motion Control Interface”, International Journal of Advanced Research in Electrical, Electronics and Instrumentation Engineering, 2015.