Low Cost FPGA Implementation of a SPI over High Speed Optical SerDes

Peter Hobden†and Saket Srivastava†
†School of Engineering, University of Lincoln, United Kingdom
16633091@students.lincoln.ac.uk, Ssrivastava@lincoln.ac.uk

Abstract – Serial Peripheral Interface (SPI) is a commonly used communication protocol that allows serial data transfer between a master and a slave device over a short distance. However, if we require just SPI over long distances currently there is no effective low-cost solution. A SerDes provides a solution to this shortcoming by sending parallel data as a serial transmission and converting it back at the receiver end. However, most of the current SerDes implementations are expensive to implement and cater to very high-speed applications, which is not the case in SPI. In this paper, we present a simple to implement and low cost SerDes solution for sending and receiving multiple SPI and GPIO lines. Our proposed solution makes use of a low cost CLPD / FPGA and is applicable for low data rate applications such as SPI. This paper investigates the simplest solution to the problem, whilst maintaining a reliable single wire / optical link. For testing, we have implemented three novel encoding schemes that all provided good results, each measured by performance against resource usage. One of these encoding schemes has shown a drop-out rate as low as 0.001% over a 24-hour period. Our proposed solution when used in conjunction with an optical fibre medium could potentially allow SPI transmission over several kilometres of distance.

Keywords— SerDes (Serializer/Deserializer), Inter Integrated Circuit (I2C), Serial Peripheral Interface (SPI), Intellectual Property (IP), System-on-Chip (SoC), Verilog, VHDL (Hardware description Language),Vivado (Xilinx's synthesis tool) .

I. INTRODUCTION

The serial peripheral interface (SPI) bus is an unbalanced or single-ended serial interface designed for short-distance communication between integrated circuits on a printed circuit board (PCB) [1] [2] [3] [4]. SPI is known as an average speed synchronous serial protocol in which, a master device (typically a controller) exchanges data with one or more slave devices. The data exchange is full-duplex and requires synchronization to an interface clock signal. Interestingly, despite the widespread usage of SPI in most of today's electronics applications, currently there is no effective low-cost solution for accessing data from an SPI bus over long distances without seriously degrading the signal integrity. This has many usages, such as providing a remote debug port for your SPI bus.

A Serializer/Deserializer (SerDes) is a pair of functional blocks commonly used in high speed communications to compensate for limited input(s)/output(s). These blocks convert data between serial data and parallel interfaces in each direction. The term "SerDes" generically refers to interfaces used in various technologies and applications. The primary use of a SerDes is to provide data transmission over a single/differential line to minimize the number of I/O pins and interconnects. In principle, the idea of converting the SPI parallel lines to a high speed SerDes, sound straight forward, however in reality there are many challenges to overcome. One of the biggest challenges is to achieve a long distance reliable link with negligible addition to the build cost. The literature on SPI protocols is extensive and the topic is classed as being old in the embedded world (it dates to the early 1980s). To the best of the author's knowledge, there are no papers that implement SPI over a low cost SerDes using an CLPD/FPGA. One recent paper highlights the need for this technology, however no details were provided regarding implementation or design [5]. Another paper makes a reference to using a coaxial transmission line [1]. We have explained this in more detail in the results section

The biggest advantage of combining the SPI with a SerDes, is the capability to transmit and receive the SPI signal over large distances. The simplest implementation would be to take all the SPI lines and connect them directly between the SerDes transceiver. This approach has the advantage that in theory you can view the SPI signals on an oscilloscope/Analyser, seeing the data lines going in and coming out at the receiver end of SerDes. However, in practice this approach is not reliable due to the following reasons: (a) Inductive effects present in long transmission lines result in *baseline wandering* [1], which causes the signal to drop out, thus resulting in a loss of communications due to the line being not DC-balanced. In simple terms, a long string of 1's or 0's would potentially create a DC-unbalanced line. AC coupling can also add to baseline wander. (b) Another challenging issue with transmitting low frequency data (such as those over SPI) is that the effective data-rates are very low (typically in the range of 10 Mbps) as compared to other long-distance protocols (E.g. Ethernet transmits in Gbps range). This wide discrepancy in data rates over the two mediums (which are being interfaced over SerDes) may cause the transmitters and receivers to go in to a sleep mode which will also disrupt data.

In our proposed solution, we have targeted challenge of extending SPI to transmit and receive data over a long distance. We address the issue of baseline wandering by implementing a novel, low cost and easy to implement/debug IP encoder block on a low cost CLPD/FPGA. Our proposed SerDes design also addresses the huge imbalance in data rates of SPI transmission (as highlighted above) and the effects of latency. The rest of the paper is as follows. Section II provides a brief introduction to SPI and SerDes technologies. Section III describes our proposed design. We discuss the results of our proposed design in Section V. The Conclusion and Summary are presented in section VI.

II. SERIAL PERIPHERAL INTERFACE AND SERDES

In this section, we briefly mention various SPI implementations, followed by an investigation in to the best methods for interfacing the SPI to a SerDes link.

A. Serial Peripheral Interface (SPI)

Most SPI interfaces are recognisable with the following data lines as shown in Figure 1.

- Clock (CLK) - An interface clock initiated by the master device to ensure synchronous data transfers.
- Master Out Slave In (MOSI) - A data line for data sent from the master to a slave (Dual/Quad has additional lines to Double/Quadrupole the transfer speed).
- Master in Slave Out (MISO) -A data line for data sent from a slave to the master. This could be shared with

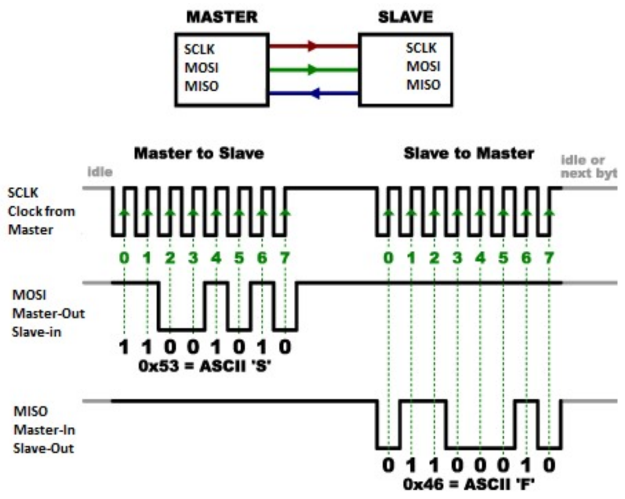

Fig. 1. SPI Block Diagram

the data line in two wire mode, if this is supported by the slave devices.

- Slave select/Chip enable (SS/CE) –-This is normally a single wire, where a low equals chip 'enable' and a high is chip 'disable', which is required for each slave device on the bus.

The SPI supports multi-slave operation. The master and slave can be the transmitter or receiver based on its mode of operation. It can receive and transmit on both the rising and falling edges of the clock independently. Normally the SPI protocol follows some design steps. To start the signals generation, the *Master* generates transaction signals with respect to the system clock and SCLK. The MOSI pin starts transferring data after to CE/SS. Whenever CE/SS is low it's able to receive data coming from master through the MOSI pin. The *Slave* transmits data through MISO pin using the shift register one by one data shift and each bit transmits every clock pulse. At the low level, the Data Register and eight-bit shift register forms the main part of the SPI system. When an SPI transfer gets a place, a bit of data gets shifted out of the SPI master's data register and then, the serial data which comes from the slave's data register in sequence is shifted into the master's data register. Therefore, by the time one SPI transmission completes, i.e. after the 16th clock cycle, the contents of the master and slave would have been swapped. Transmissions often consist of 8-bit words.

To begin communication, the bus master configures the clock, using a frequency supported by the slave device, typically up to a few MHz's. There's no defined limit, except that you can only run as fast as the maximum clock rate supported by the slowest slave device on your bus. This is normally 50 MHz's due to the limits of TTL switching speeds. This clock rate is important as we must make sure our SerDes encoding scheme supports the lowest to the highest clock rates.

B. Communication Protocols for SPI

In the world of serial data communication, SPI interface is considered as a small communication protocol and as such doesn't have a standard [3] [6]. USB, Ethernet and RS232 Serial are meant for "outside the box communications" and data transfer in the whole system while serial peripheral interface communication between integrated circuits is classed as little or middle data transfer rates. We have ascertained that an encoded scheme is required, which shall convert our SPIs lines to parallel encoding lines that can be decoded and reconstructed by our receiver, without adding any time latency. There are many SerDes protocols in everyday use, such as PCIe 4/3/2/1, USB 3.1 Gen 2 and Gen 1, 10G-KR, SATA3/2/1, SFP+, RXAUI, XAUI, QSGMII and SGMII. All these protocols are quite complex and require a large overhead to implement and all add latency. For non-optical applications, there is a good case to use the same SerDes protocols for SPI instead of developing new ones. A good example of this would be the how companies such as FTDI make a chip set that converts USB to SPI and vice versa [7]. In telecommunications, a similar problem existed to ours, and this was solved using 8b/10b encoding scheme [8] [9] [10]. This scheme consisted of 8-bit message words having two corresponding 10-bit code words. One of these has positive disparity (more 1's than 0's) and the other has negative disparity (more 0's than 1's). When encoding you keep track of the running disparity. If the running disparity is positive and the next input octet gives you a choice of code words, you then pick the one with negative disparity, and vice versa. In this work we are looking at developing the simplest protocol that could be implemented when resources are limited, so the

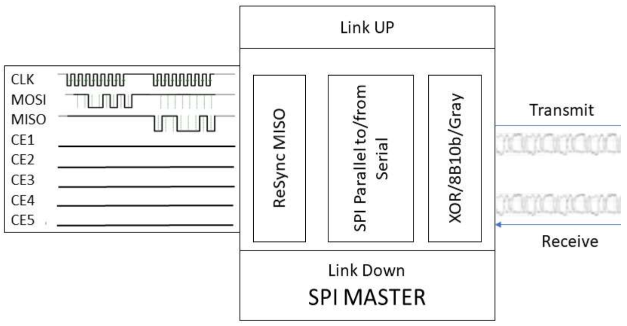

Fig 2. FPGA implementation of the SPI Master showing different component blocks

encoding can run on low cost CLPD/FPGA devices, using the minimal of resources and reducing latency. We have implemented three different encoding schemes to test the proposed design as shown in Figure 2.

1) 8b/10b scheme

Xilinx provides 8b/10b a decoder IP block for their high-end processors (Virtex-2 onwards). Opencores also provide a generic synthesizable VHDL, which provides two separate cores for encoding and decoding byte data according to the 8b/10b protocol that closely follows the original.

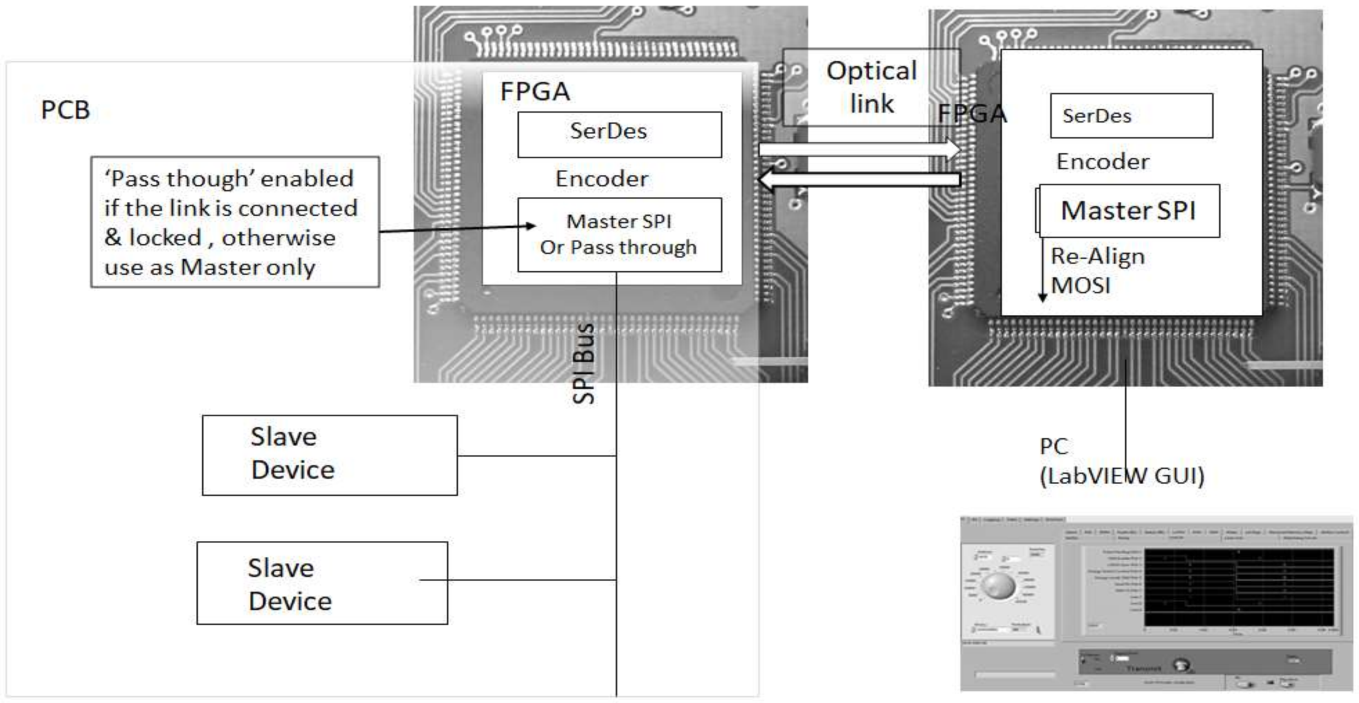

Fig. 3. Block diagram showing the SerDes connected on the SPI bus

2) XOR scheme

An alternative method to the one discussed above is to make sure that the data is continuously changing about the clock. For example, the data lines can be XORed against a slower running clock. This will not create a perfectly DC balanced signal, but it is the easier method to implement in VHDL/Verilog, taking little resource overhead, so it can run comfortably on a low cost CLPD.

3) Gray code scheme

The XORed method can be further advanced by using Gray codes. This is known as reflected binary code (RBC), also referred to just as reflected binary (RB) or Gray code [11]. The RBC is an ordering of the binary numeral system such that two successive values differ in only one bit (binary digit). The RBC was originally designed to prevent spurious output from electromechanical switches. Today, Gray codes are widely used to facilitate error correction in digital communications such as digital terrestrial and some cable television systems. The resource requirement for Grey is greater than the XORed method, but still classed as low.

C. SerDes Tranceivers on FPGA

As mentioned earlier, a Serializer/Deserializer (SerDes) is a pair of functional blocks commonly used for high speed communications while compensating for limited input(s)/output(s). Except for some higher end FPGA/SoC platforms, most other manufacturers do not provide a high-speed IP SerDes block making use of the on-chip resources, for the new JESD204 standard, mainly due to the cost and resource constraints. JESD204 is a high-speed serial interface for connecting data converters (ADCs and DACs) to logic devices. Revision B of the standard supports serial data rates up to 12.5 Gbps and ensures repeatable, deterministic latency on the JESD204 link. As the speed and resolution of converters continues to increase, the JESD204B interface has become ever more common in high-speed converters and integrated RF transceivers. In addition, flexible SerDes designs in high end FPGAs have naturally started to replace the traditional parallel LVDS/CMOS interface to converters and are used to implement the JESD204B physical layer [12]. The JESD204B specification defines four key layers that implement the protocol data stream. The transport layer maps the conversion between samples and framed, unscrambled octets. The optional scrambling layer scrambles and descrambles the octets, spreading the spectral peaks to reduce EMI. The data-link layer handles link synchronization, setup, and maintenance, and encodes/decodes the optionally scrambled octets to/from 10-bit characters.

D. Optical SerDes prerequisites

For a SerDes to work, on the receiver side, a clock must be recovered from the data stream before accurate byte/word alignment can occur. The clock recovery block can only recover the clock and data if the input data stream contains adequate data eye and the average DC component is zero. Our encoding schemes need to be mindful of this. One other reason that could cause the interface to malfunction is the lack of synchronization due to large propagation delays. Over an optical link, this should be small but if the line length is too long, this could create an issue as the MISO and MOSI could potentially become out of sync. The solution to this initially is to run the SPI at a lower clock rate. A novel approach to the solution used in this paper is to include in scheme an automatic adjustment when re-creating the MISO line at the receiver end, by re-syncing the MISO with the SPI clock line. This is the advantage with implementing our own scheme as opposed to a fixed hardware solution.

III. PROPOSED SERDES IMPEMENTATION ON AN CLPD/FPGA

This section discusses the various options for implementing SPI with SerDes on a low cost CLPD / FPGAs. Unlike microcontrollers, FPGAs are flexible and don't rely on hardware blocks. For the SPI implementation, IP blocks such as the 'Quad SPI' can be used, or writing a HDL module or even creating a 'Bit Bang' C variant running under the Xilinx Microblaze/Arm/Lattice Mico8. In the case of some FPGAs like the Zynq and Ultrascale, the SPI can also be implemented in hardware. This is then connected to the SerDes block, for converting the SPI lines in to a single serial line. If the SerDes link is down/not connected, the SPI block becomes the master controller. If the link is up, then the remote host becomes the master. This can be implemented on a single CLPD/FPGA or two FPGAs (one for SerDes, the other for the Master SPI) as shown in Figure 3.

Once you've implemented the SPI, this then needs interfacing with our SerDes block. The standard interface is the AXI4-Lite / AXI4 interface and in the case of Zynq, the PLB (Processor Local Bus). One alternative to the AXI4-Lite / AXI4 interface is the wishbone bus [13] as used by 'Opencores' and used by other FPGAs such as the Mach X02 from Lattice. In the Vivado and Lattice Diamond design environments, digital logic can be defined using a block diagram rather than writing traditional HDL code. Behind the scenes, this generates the HDL code for you. These are called *IP blocks*. Xilinx provide a soft SPI IP block called the 'Quad SPI'. The reason why it is called the quad I/O, is that the four-bit data bus interface improves throughput by four times. The additional lines are primarily used where performance is critical, such as for Flash memory devices. One of the advantages is that it also operates in a "legacy mode" acting as a normal SPI controller, so it can be used to provide a normal 'soft SPI'. The SPI should be connected to the AXI_LITE bus and optionally to an interrupt.

Devices like the Zynq have two hardware SPI controllers, using the MIO (Multiplexed Input Output) and one quad SPI, which is dedicated for reading/writing to flash memory. In the case of the MIO pins, the user constraints should not be present as these pins are fixed by the hardware. This configuration is then exported to software developers' kit (SDK), where control codes can be implemented. For development, Microprocessor cores could be used for extra debugging to provide a low data rate serial port for say connecting our LabVIEW GUI tool. For a production version, it is not recommended to have this function as it will increase the cost.

In our proposed solution, the idea is that the CLPD/FPGA is divided into two blocks, one for handling the SPI and the other for handling the SerDes including using our selected encoding scheme. A novel part of the system, is to add a block to re-synchronize the MISO lines after the decoding, allowing for a long-distance connection. On the remote PCB, if the SerDes link is down, the CLPD/FPGA shall operate in 'Master mode' using the methods that we have discussed. If the SerDes is connected, the bus shall then automatically switch to using the remote connection as the Master (shown in Figure 3).

One advantage with having such a flexible system is that if we ran out of parallel lines, we could create some virtual additional lines just by using a clock edge and toggling to a new set. The downside to this is that the potential maximum data rate is then halved.

IV. HARDWARE INTERFACING

In our proposed implementation of SPI over SerDes, we have opted to interface the SerDes over an Optical fibre. In our example, we have implemented the (TI) SN65LV1023A Transmitter together with the (TI) SN65LV1224B Receiver, which is in turn connected to our CLPD/FPGA. Each device

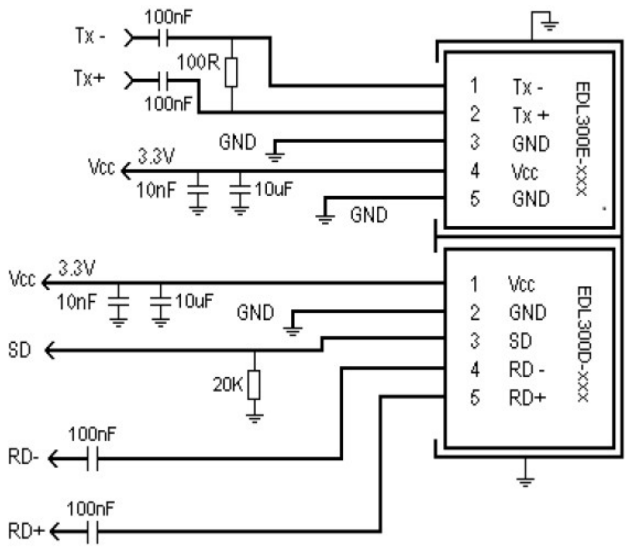

Fig. 4. LVDS SerDes Transmitter/ Receiver

works from a crystal input from which it multiplies by 12. It then reads in 10 bits of data and transfers these over the serial interface at the clock x12 speed, so for 12.5 MHz it is transmitted at 150 Mbps. This is then connected to two 'Firecomm EDL300k-120 optical diodes as shown in Figure 4. Data can then be transmitted/received via an Optical fibre cable to a duplicate setup. In our example we have made use of an optical fibre link for long distance transmission, however these could be replaced by other communication mediums (E.g. a coaxial cable). At 150 Mbps, the connection between the transceiver and the optical diodes is critical and as a result we need to consider transmission line theory and pay careful consideration to the PCB board layout. In our implementation, we found that the voltage levels between the transceivers and the optical devices were compatible, so we were left with another challenge to ascertain whether use AC or DC coupling for the interface.

The question regarding the use of AC or DC coupling for the interface line between the transceivers and optical devices depends on the presence or absence of DC offset (which is generally present during data transmission). If the mean amplitude of the receive/transmit signals is zero, there is no DC offset. A waveform without a DC component is known as a DC-balanced or DC-free waveform. Therefore, our default position is always to recommend AC coupling as it prevents DC power flow from IC-to-IC. DC power flow can cause destructive effects during unexpected power surges or failure of DC regulation. Our second major reason for recommending AC coupling is that while many IC's claim to have an LVDS interface this is not always the case. Our experience is that there is a lot of confusion in datasheets between LVDS and LVPECL. In our case DC coupling is obviously an easy step as both IC's are 100% LVDS. However, for our general recommendation, the safe option is always AC coupling. To

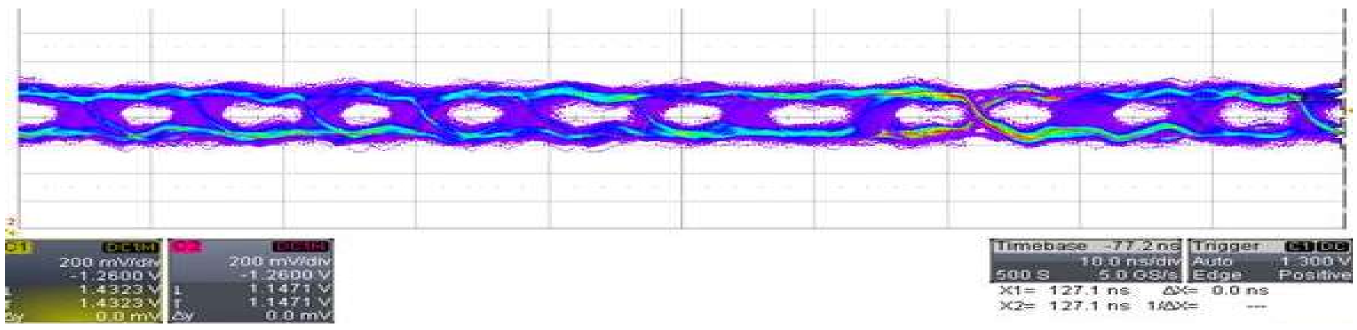

Fig. 5. Eye diagram of the SerDes link produced using the LeCroy High Frequency Oscilloscope

add AC coupling capacitors (ideally 100nF), then place them in series with the TD+, TX-, RD+ and RD- lines. One needs to make sure that the capacitors are as close as possible to the transceiver module.

V. RESULTS

To fully test the SPI signal transmission over optical SerDes, we ran the link for 24 hours using each one of the three proposed encoding schemes proposed in section II.B. During the test, random SPI traffic was generated. The received results were compared with the transmitted data and the number of errors recorded. Note that the SPI data clock was randomly set from 10–50MHz. A LabVIEW GUI was written to record parity and locking errors. The LabVIEW GUI also provided a graphical logic analyser display for viewing the encoded and decoded signals and the result recorded result were recorded in Table 1.

For debugging with a scope, the 'non-encoded' method seems most logical and you can easily view the transmitted and received signals, but it is unreliable, and the signal dropped out regularly. When we tested the design using our proposed encoding schemes (XOR, Grey codes and 8b/10b / 8B10B), the line stability increased manifold. However, incorporating these encoding methods make debugging more challenging as the SPI signals are no longer visible on a scope as regular SPI signals.

Amongst the three proposed encoding methods, 8b/10b / 8B10B is the most reliable with just 1 dropout in every 10,000 transmissions, as it creates a more DC balanced signal. However, this scheme is not so suitable for running on a low cost CLPD due to the high recourse requirement. The XOR and Grey Code schemes although less reliable, are much easier to implement and debug with very low resource requirements. Overall, among the three proposed schemes, the Grey Code encoding scheme provides the most optimum solution for long distance SPI signal communication.

As a further check on the links integrity the most common industry standard is to produce an eye diagram, which highlights the condition of the transmitted signal [14]. An eye diagram of the links traffic was produced using a LeCroy high frequency oscilloscope as shown in Figure 5. The SerDes system has been found to operate in rather a specific way and should not be used in ways that can compromise these findings. The initiation sequence of SerDes needs a sync pulse on the sync line and then the RX/LOCK line to be observed to manage the link state, making sure that the link locks correctly. One interesting observation, if the data on the data lines are invalid and not using one of our encoded schemes, the link can quickly go unstable and fault from at least the first 5 bytes sent. If the /RXLOCK signal is not asserted then the link is unstable and should not be used, a fault should be flagged here. If the link is synchronised, then it should continue to use the encoded data through the link otherwise the link could still fail.

TABLE 1. COMPARATIVE PERFORMANCE OF THE THREE PROPOSED ECODING SCHEMES.

FPGA Scheme	**Results**	
	Drop Out %	***Resources***
No encoding	~100% (did not work)	5%
No encoding with a clock line	~75.01%	7%
XOR	~6.95%	8%
Grey Codes	~3.5%	9%
8b/10b / 8B10B	.001%	24%

VI. CONCLUSION

With this paper we have demonstrated that you can successfully implement a single wire low cost SerDes link for sending/receiving SPI and GPIO data, using a low cost CLPD/FPGA, taking the minimum of the chip's resources and reducing any issues latency. We have also demonstrated that if latency is an issue over a long distance, we can successfully re-sync the MISO line using our novel scheme, which isn't possible with any hardware solutions.

REFERENCES

[1] T. Kugelstadt, "Extending the SPI bus for long-distance communication," *Analog Applications Journal.*

[2] MOTOROLA INC., "SPI Block Guide V03.06 , Document number S12SPIV3/D," *Document number S12SPIV3/D ,* no. 04, 2003.

[3] F. March. Leens, "An Introduction to SPI Protocols - Motorola.," 2003.

[4] F.Leens, "An Introduction to I2C and SPI Protocols," *IEEE Instrumentation & Measurement Magazine,* pp. 8-13, 2009.

[5] Mike Dewey ,Jim Kent, "Implementing Digital Interfaces with User Programmable FPGAs," *IEEE,* 2017.

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

[7] FTDI, "SPI to USB," [Online]. Available: http://www.ftdichip.com/FTDrivers.htm.

[8] A. X. Widmer, P. A. Franaszek, "A DC-Balanced, Partitioned-Block, 8B/10B Transmission Code," *IBM Journal of Research and Development ,* vol. 27, no. 5,

1983.

[9] K. Odaka, "Digital Audio Tape". US Patent 4,456,905, 1984.

[10] K. S. Immink, "Digital Audio Tape". US Patent 4,620,311, 1986 .

[11] F. Gray, "Pulse code communication". US Patent 2,632,058, 17 03 1953.

[12] T. Hill, "Comprehensive JESD204B Solution Accelerates and Simplifies Development," *White Paper: All Programmable FPGAs and SoCs,* vol. WP446 , no. (v1.0.1), 2014.

[13] Opencores, "Specification for the "WISHBONE System-on-Chip (SoC) Interconnection Architecture for Portable IP Cores," 7 7 2002. [Online]. Available: https://cdn.opencores.org/downloads/wbspec_b3.pdf.

[14] ON Semiconductor, "Understanding Data Eye Diagram Methodology for Analyzing High Speed Digital Signals," June, 2015 − Rev. 1.