Contents lists available at ScienceDirect

e-Prime - Advances in Electrical Engineering, Electronics and Energy

journal homepage: www.elsevier.com/locate/prime

High bandwidth data and image transmission using a scalable link model with integrated real-time data compression

Rand B. Mohammed, Ph.D.[a,*], Roelof van Silfhout, Ph.D.[b]

[a] Mechatronics Engineering Department, Tishk International University, Erbil, Iraq
[b] Department of Electrical and Electronic Engineering, University of Manchester, Manchester, United Kingdom

A R T I C L E   I N F O

Keywords:
Real-time
Transceiver system
FPGA
Embedded system

A B S T R A C T

Our society increasingly relies on the transmission and reception of vast amounts of data using serial connections featuring ever-increasing bit rates. In imaging systems, for example, the frame rate achievable is often limited by the serial link between a camera and host even when modern serial buses with the highest bit rates are used. This paper reports a scalable link system with a bandwidth and interface standard that can easily be adapted to suit a particular application. The scalable serial link approach has been extended with lossless data compression with the aim of further increasing dataflow at a given bit rate. The compression method is integrated into the scalable transceivers providing a comprehensive solution for optimal data transmission over a variety of different interfaces. The system is fully implemented on an FPGA using a fully hardware based system. A Terasic DE4 board was used for implementing and testing the system using Quartus-II software and tools for design and debugging. The impact of compressing the image and carrying the compressed data through parallel lines is similar to the impact of compressed the same image inside a single core with a higher compression ratio, in this system between 7.5 and 126.8.

1. Introduction

Transferring high resolution multi-spectral images between a camera or medical diagnosis device (e.g. CAT, MRI) and an offline processing system (host) can be particularly challenging, especially if transfer times must be shorter or equal to the frame time of images so as not to create bottlenecks in the image data transfer. For the majority of modern camera systems the transfer link speed to a host is the limiting factor. One way to improve the effective frame rate as received by a host is to use image compression inside the camera. Provided a fast image compression system is available that provides compressed images within the time it takes to capture a new frame such an approach should increase the effective frame rate when the image data transfer is the limiting factor. Some companies (e.g. Dectris https://www.dectris.com/), have used an approach that transmits images to clients over multiple transceiver links. The transceiver devices are based on SerDes technology, which matches different transceiver technologies such as the Ethernet, the PCI Express [1–3] and the ESATA/SAS standards [4–8]. With these systems, data is carried over the transceiver links from the source, without any changes in their format and without applying any form of image processing. For example, an x-ray detector with 2070×2167 pixels where each pixel is digitized with 32-bits per pixel that captures images with a frame rate of 750 Hz produces an uncompressed data rate of 108 Gbps. Such data rates are well in excess of any single serial link that is currently available; most links are limited to data rates of 1–10 Gb/s. Because of transfer protocol overhead the effective image data rate is usually significantly below the advertised bit rate resulting in a data transfer bottleneck.

In this paper we discuss two approaches to address the resulting transfer bottleneck. On the one hand, data is compressed using an optimized lossless compression algorithm whilst on the other hand bandwidth is increased by the use of a flexible or scalable data link approach. In order to handle the high data rate, the system cannot be implemented by using a traditional sequential microprocessor based system with software. Instead, we propose to create a scalable multi-link system based logic modules that are implemented on a Field Programmable Gate Array logic (FPGA). A full system consists of an embedded system implemented on an FPGA on either side of a variable set of transfer links (see Fig. 1). Each system handles image (de)compression algorithms, data (de)serialization and a dedicated transfer protocol.

The architecture of a traditional system is implemented with a central processing unit (CPU) that is operated by software and interfaced to hardware. The TCP/IP protocol is one of the most common methods of creating a serial link that uses an Ethernet physical layer which typically consists of 1–4 twisted wire pair connections. When a project is built as an embedded system and based on TCP/IP as a standard protocol, it is called an embedded TCP/IP protocol. The embedded TCP/IP

* Corresponding author at: Mechatronics Engineering, Tishk International University, Erbil, Kurdistan, Iraq.
E-mail addresses: rand.basil@tiu.edu.iq (R.B. Mohammed), r.vansilfhout@manchester.ac.uk (R. van Silfhout).

https://doi.org/10.1016/j.prime.2021.100017
Received 9 September 2021; Accepted 28 October 2021

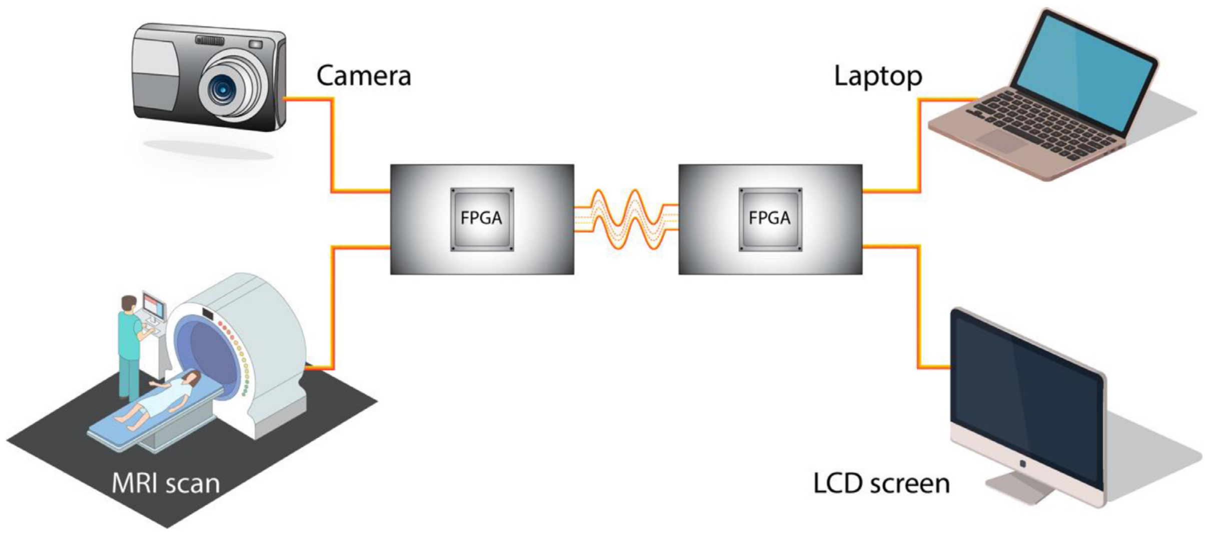

Fig 1. An example of a scalable multi-link architecture which connects different imaging devices to client devices [12].

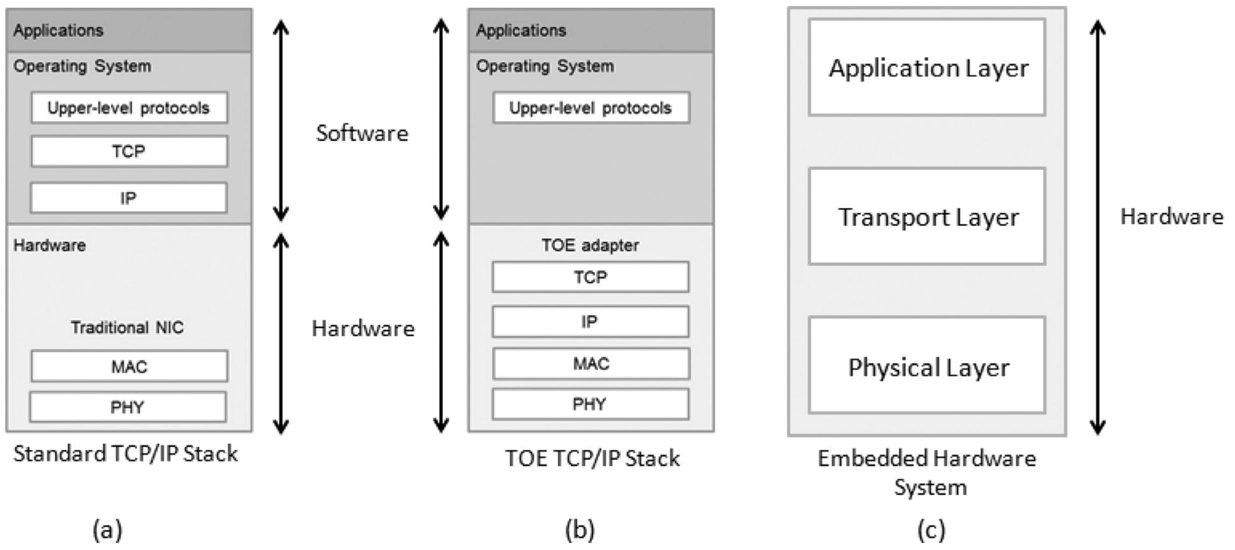

Fig 2. The architecture of embedded software and hardware systems (a) and (b) [13] and an embedded hardware system (c) [12].

has a structure of a hierarchical protocol that is similar to the TCP/IP structure in the PC [9]. The main characteristics are real time, flexibility and simplicity [10]. The embedded TCP/IP system is usually built as partially software and partially hardware; the software part consists of the CPU that controls the whole system processor with a slow data-rate, which can cause a CPU bottleneck. In the past few years, the attractive solution to this problem, viz. The TCP/IP Offload Engine (TOE) emerged to reduce the CPU overhead in order to improve the network performance and increase the throughput of network connection [11]. The increase in the Ethernet speed can support between 10 Mbps and 10 Gbps. Fig. 2b shows the architecture for the TOE based on the traditional TCP/IP. The architecture of the second type of embedded system, which is implemented as hardware without a software part (no CPU) is shown in Fig. 2c.

The TCP/IP Offload Engine (TOE) was developed by different researchers. Zhong-Zhen [13] and Hashimoto [14] used the TOE for the Gigabit Ethernet system, Zhong-Zhen [13] designed and implemented the TOE as a solution to the TCP/IP CPU bottleneck. The system was implemented as an embedded system inside the FPGA device, which was connected to the desktop PC via a Gigabit Ethernet switch. The system was tested with a packet size equal to 1.5 K-bytes, the transmitted data-rate was 189.23 Mbps while the receiving data-rate was between 239.34 Mbps and 296.32 Mbps. The transmitting data-rate was made slower than the receiving data-rate since the transmitted data requires more CPU cycles to process the TCP/IP encapsulation. Chang [15] developed two TOE virtualization architectures in order to enable multiple operating systems communicate with real computers, and share the same physical machine. Yong [16], on the other hand, used the TOE for the 10 Gbps, by dividing the TOE system architecture into eight blocks and using four XAUI interfaces to transmit and receive the frames. The system was implemented on a Stratix IV GX device, and the transmission rates were up to 40 Gbps as sender and 4 Gbps as receiver.

2. Methodology

The Scalable Link Model with Compression (SLMC) is a new reference model used to carry real-time images or data between cameras/detectors and clients, a source and a destination, at high transceiver speeds. The SLMC is designed to support different types of application for instance it can be used in remote sensing camera applications that required data miniaturization, high speed data transmission, and support real-time [17].

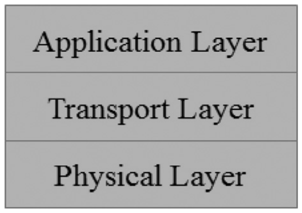

Fig 3. The SLMC architecture.

The SLMC divides the transmitter and receiver systems into three layers, i.e. application, transport and physical, as shown in Fig. 3. Each layer has a specific task, and in combination with the others it provides a full transceiver system. The physical layer interfaces with the actual physical media, the application layer specifies and handles the protocol of transmitting/receiving data whereas the transport layer controls the flow of data between the previous two layers.

One of the main objectives of the SLMC is to overcome the complexity involved in using other reference models, such as TCP/IP, and their required protocols as well as the limitations of using a single transceiver link. A second objective is to increase the effective transceiver speed by using a multiple links in parallel at the physical layer level, for both the transmitter and the receiver. A third objective is to reduce the data volume by using a compression method at the transmission side and a decompression method at the receiving side. A lossless compression method is used as a part of the application layer. The SLMC allows for the control of more than one receiver application at a time, each being placed into an individual domain, which requires applying a few changes to the SLMC receiver side, especially of the transport and application layers.

2.1. *The application layer*

The application layer is the top layer of the SLMC which provides the system with the ability to access an application device in order to exchange data. The SLMC has a full architecture for the transmitter and receiver sides, and different processors are applied to the data.

On the transmitter side, as shown in Fig. 4, the application layer consists of several sub-layers, some of which are physical devices, such as the source of data (camera) with an external memory device, and others are implemented as an embedded system inside the FPGA device. The camera generates real-time data which are stored temporarily inside the external memory, and which then interface with the embedded system inside the device. The embedded system part of the application layer consists of four sub-layers (see Fig. 4) which carry the data before the embedded system interfaces with the memory in order to read the data and process it accordingly. The first sub-layer provides a connection between the external part of the application layer and the rest of the system, the second sub-layer controls reading data from the memory controller, the third sub-layer divides long packets read from the device into multiple sub-packets, and the final sub-layer compresses data on the transmitter side based on the lossless compression method called RLCM (reduced lossless compression method) [18].

However, on the receiver side, as shown in Fig. 4, the application layer is placed at the final stage of the system and consists of several sub-layers, some of which are physical devices such as the destination device with the external memory device, while the others are implemented as an embedded system. The external memory device interfaces with the embedded system inside the device and is used to store the data received in order to forward them to a destination application such as the Vision System. The embedded system part of the receiver application layer consists of several sub-layers. The first sub-layer de-compresses data that has been compressed on the transmitter side, based on the same compression method. The second sub-layer combines sub-packets of data in order to generate a full packet of the original width. The third sub-layer is responsible for controlling writing data into the final sub-layer (the memory controller), which provides a connection with the external memory device and is used to monitor data flowing between the embedded system and the memory device.

The available technique for compressing the image is by dividing the image into eight sub-images called packets, 8K-pixels each. Each packet is compressed inside a single compression core, which means there are eight single systems all connected in parallel and share the same source of data. Carrying data from each single core is based on the pipelined technique. When compressing data by using parallel cores making the execution time eight times shorter than when compressing the image using single core. However, the execution time at each core can be either the compressed time for each core or the transmitted time, depending on which one is longer. In this case the delay will be the time required to compress the first row from the sub-image when its size is 256-pixels. The main reason behind choosing the size of the data processed each time to be equal to the row size (i.e. 256 pixels) is that the compression method was designed to compress 256 pixels at a time. In compressing the image inside one core, the system takes 256 processor-cycles to complete the compression, while the system takes 32 processor-cycles to compress the same image inside eight parallel cores; when the processor-cycle is the time to compress each row from the image.

2.2. *The transport layer*

The transport layer is implemented at both the transmitter and the receiver sides, between the application and physical layers. It has two main concepts. First, it carries data on the transmitter side from the application layer to the physical layer. Secondly, it applies a process to the data in order to make it suitable for carrying over physical media through transceiver devices. These processes are changeable based on the transceiver speed supported by each device at the physical layer, which can be either 1 Gbps or 3.125 Gbps.

One of the available transceiver speeds that can be used for a single device is 1 Gbps, which can be supported by many transceiver protocols such as low-voltage signal (LVS) and Gigabit Ethernet (GIGE). The designs of the transport layer and its cores for both the transmitter and the receiver sides have a similar structure and a similar means of implementation. On the transmitter side, each individual core on the transport layer reads data from the application layer and writes it inside the physical layer. The width for reading data is equal to 16 bits, while the width for writing data into the physical layer is 8 bits. To help match data between these two layers, a data controller and a divider, both of which are implemented as embedded systems. The data controller is used to manage data flowing between the layers, while the data divider is used to divide each frame into two parts, the MSB and the LSB, and forward them to the next layer, i.e. the physical layer. On the receiver side, the opposite process is applied, with each core in the transport layer carrying data from the physical layer to the application layer. Each core is implemented as two combiner and data controller sub-layers, each with two interface ports – one with the physical layer, with a width equal to 8 bits, and the other one with the application layer, with a width of 16 bits. The combiner reads the data from the physical layer with a byte width, and then it requires two clocks to generate the full frame with a 16-bit width before sending it to the data controller. The MSB is read with the first clock and the LSB with the second clock. After combining two bytes, the combiner will send the data, at its original width, to the next sub-layer data controller in order to forward the packet to the application layer.

However, for the transceiver system that supports 3.125 Gbps as a maximum transceiver speed. Unlike the transport layer, for 1 Gbps systems, the data width for both sides is equal to 16 bits, in which case the transport layer just needs to pass data between the two layers, without applying any extra processes. These cores are called 'data controllers' and are implemented as an embedded system and connected in parallel

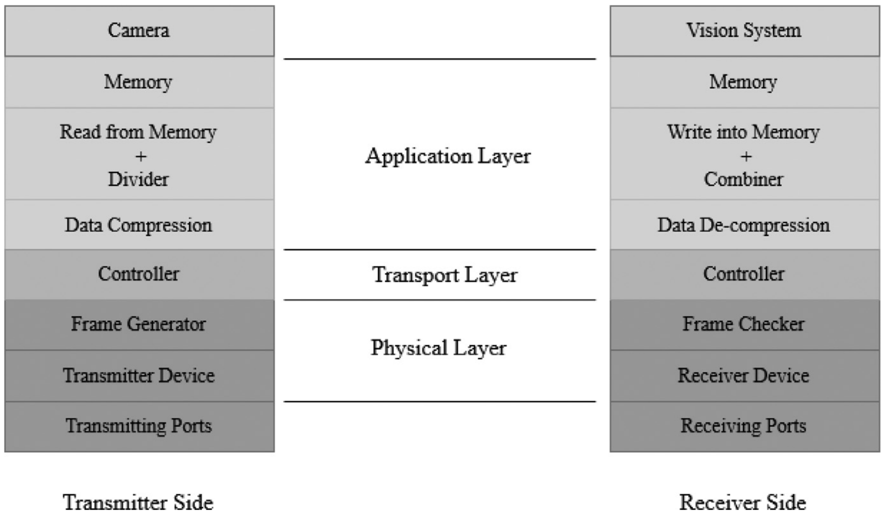

Fig 4. A diagram of the proposed system.

at each side. All the data controller cores are triggered with the same clock in order to synchronize data movement.

2.3. The physical layer

The physical layer is one of the most important parts of the SLMC system. The name of the layer defines the real purpose, which is based mainly on connecting one or two applications in different systems through physical media, in order to exchange data. The physical layer has two interface ports, one of which is connected with the rest of the embedded system inside the FPGA while the other one exports outside the FPGA in order to connect to the transceiver ports and devices. The physical layer consists of two sub-layers (see Fig. 4), both of which are implemented as an embedded system, and the bottom sub-layer interfaces the physical medium.

Based on its main structure, the SLMC is designed to support various transceiver speeds, protocols and devices, depending on the available input/output (IOB) ports for the main FPGA device that is used to implement the embedded system. The SLMC physical layer consists of multiple transceiver systems connected in parallel at both the transmitter and the receiver sides. Generally, each transceiver system can support either 1 Gbps or 3.125 Gbps as a single transceiver speed. These transceiver speeds are provided by different transceiver protocols in order to match different physical media and transceiver devices.

Data transmission over parallel nodes is important, especially for those systems requiring high data transmission rates. In SLMC systems, different transceiver protocols are used, some of which (e.g. the Serial-Rapid-IO) are already based on the internal parallel structure and are modified to be used with the external parallel structure. Other protocols, such as the LVS and the GIGE, which are designed to support a single transceiver system only, are modified to support a parallel structure.

2.3.1. Physical architecture for the LVS protocol

The LVS support link speeds up to 1 Gbps for relatively short distances, and it is based on the serialiser and de-serialiser (SerDes) operations at both the transmitter and the receiver sides. High-speed serial transceiver protocols are currently used in most development applications which require high speeds with less pins. In this project one of the SLMC architecture models consists of eight single transceiver systems connected in parallel at the physical layer, each supporting the LVS as a single transceiver protocol. The LVS transceiver protocol has two main concepts. First, the transmitter side converts parallel low-speed data into high-speed serial data, while the receiver side converts high-speed serial data into low-speed parallel data. Secondly, the output signal from the LVS system at the transceiver side and the input signal into the LVS system on the receiver side are both differential signals and are therefore more stable than single-ended signals in order to avoid any noise while transferring data.

2.3.2. Physical architecture for the GIGE protocol

The SLMC structures also supports the GIGE as a transceiver protocol. Its physical layer consists of eight single transceiver systems connected in parallel, and each system carries data based on the GIGE protocol. The transceiver system consists of transmitter and receiver systems, which means the SLMC offers the ability to work as a transmitter system or a receiver system, or both. For transmitting, the single transceiver system is divided into two sub-layers, a frame generator and a transmitter device. The transmitter device for the GIGE transmitter protocol is called a GIGE transmitter device (GIGE-TD) which consists of two cores: the PCS and the PMA. The frame generator reads data from the transport layer and generates an input frame for GIGE-TD. It has two interface ports, the first of which interfaces with the transport layer connected internally inside the embedded system, while the other one connects to the GIGE-TD device which consists of two signal buses: data and control, which are connected internally inside the system. The frame generator reads a packet of data up to 1024 bytes in size from the transport layer, then adds a 3-byte information header file to each packet of data and finally sends it to the GIGE-TD over a bus width of 8 bits with a 1-bit control signal. The control signal is used to define the type of incoming data. If it is equal to 1, this means that the data being carried are at the same time control data; otherwise, if it is equal to 0, the data carried over the data bus are original.

The next sub-layer in the transceiver system is called the ‘transmitter device’, which consists of two cores, the PCS and the PMA, with a PMD in some cases. The PCS core is connected between the frame generator and the PMA core. The PCS core is used to encode data and forward them to the PMA core. The 8/10 encoder is mainly used with this system. The output code from the PCS is low-speed parallel data which are sent to the next core, i.e. the PMA, in order to convert them to high-speed data. The PMA core includes an SER that is used to convert low-speed parallel data into high-speed serial data, while the SER core has two input clocks, as shown in Fig. 5. The low-speed clock connects at the PCS side and is used to synchronize parallel data, while the high-speed clock connects at the SER output core in order to trigger serial data. The low-speed clock is 125 MHz used to clock the full system, while the value of the high-speed clock is 1 GHz.

However, for receiving data, the transceiver system consists of two sub-layers: a receiver device and a frame checker. The receiver device consists of a PMA and a PCS core, with a PMD in some cases. The PMA works mainly as a Deserialiser (DES) that receives high-speed serial data,

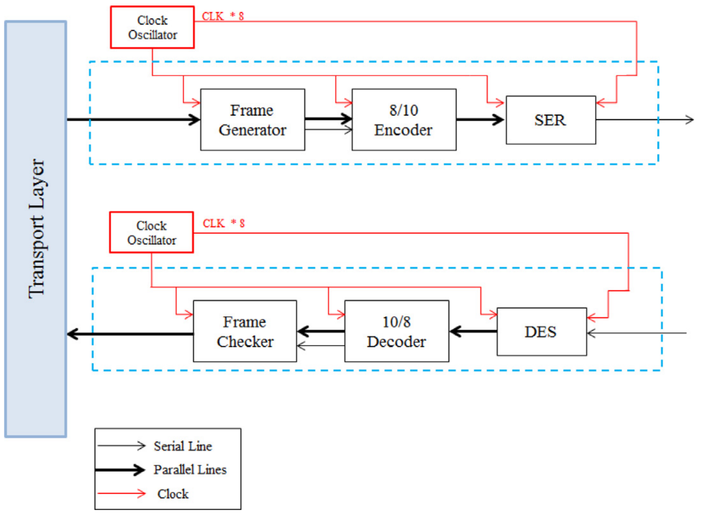

Fig 5. The interior design of a single transceiver system based on the GIGE transceiver protocol.

either directly from the physical medium or from the PMD core and converts it to low-speed parallel data. This core has two input clocks, as shown in Fig. 5. A high-speed clock equal to 1 GHz connects at the serial data side, and a low-speed clock equal to 125 MHz connects at the parallel data side. The low-speed parallel data generated by the PMA are forwarded to the PCS core, which consists of a decoder. The decoding methods applied on receipt of the data must be the same methods applied to the data before transmission; in this case the 10/8 decoder is used. The PCS generates 8-bit data with a 1-bit control signal and forwards them to the frame checker. The frame checker then reads the data with its control signal. If the control is equal to 1, the incoming data are control data; otherwise, if it is equal to 0, they are original data. The next step involves removing the header file that is added on the transmitter side and forwarding the data on to the transport layer.

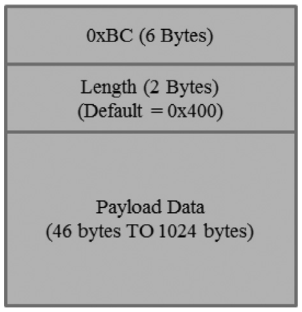

Fig 6. The Rapid-IO header file

2.3.3. *Physical architecture for the Rapid-IO protocol*

Rapid-IO is an embedded transceiver protocol used in many industrial applications. For transmitting data, it converts low-speed parallel data into high-speed serial data, and vice versa for receiving data. The SLMC's physical layer consists of multiple single transceiver systems connected in parallel, and each transceiver system is divided into two sub-layers. For the transmitter side, we have a frame generator with a transmitter device, while the receiver side has a frame checker and a receiver device. Both the frame generator and the frame checker cores are created as a customize IP-blocks for this system while the transmitter and the receiver devices used are provided by Altera in Quartus-II library.

On the transmitter side, the Frame Generator (FG) is the first sub-layer in the transceiver system on the transmitter side. It is implemented as an embedded system inside the system that connects between the transmitter device and one part of the transport layer. The process inside this core involves reading data from one part of the transport layer equal to 16-bit widths and various lengths (between 46 bytes and 1024 bytes), then adding a header file to each data packet. The FG has two interface ports, one with the transport layer over a 16-bit data-bus, while the other is with the PCS core on the transmitter side with two buses, the control bus and the data bus. The data bus, with a 16-bit signal, carries data to the PCS, while the 2-bit control bus is used to inform the PCS of the type of incoming data. If they are equal to 0, the information being carried over the data-bus is either a length or a payload-data; however, if the value of the control signal is 1, the information carried over the data bus is control data. Besides connecting two layers, the other main purpose of the FG is to generate the data in a format acceptable to the next core, i.e. the PCS core, by adding a header file to the original data packet (see Fig. 6). The header is used to align the PCS core with the coming data and it is assigned to 0xBCBC for at least three clock cycles. The next two bytes are used to specify the length of the packets in bytes, and the value of the length varies according to the length of the coming packet, and it is assigned by the previous core.

On the receiver side, the Frame Checker (FC) is a second sub-layer in the transceiver system on the receiver side which is built as an embedded system inside the main device. The FC connects between the PCS core and the transport layer. The FC receives data from the PCS core over two buses, the data bus and the control bus, the control values of which define the status of the data carried over the data bus. If it equals 1, the data bus carries either a control or an error message, but if the value is 0, the data bus carries either the length or the payload-data. The FC receives only the payload-data with the length, removes the header data that are added to the packet at FG, and then forwards the data to the transport layer. The PCS core is the first sub-layer in the transceiver system and is implemented inside the system as a part of the physical layer. It works as an encoder on the transmitter side and as a decoder on the receiver side. The PMA is the final stage in the physical layer, and it connects the embedded system with the physical medium on both the transmitter and the receiver sides. The PMA core is based on the SerDes process. For transmissions, the PMA works as an SER by converting low-speed parallel data into high-speed serial data, in order to send them out to the physical medium. On the receiver side, the PMA works as a DES by converting high-speed serial data received from the physical medium into low-speed parallel data in order to forward them to the PCS core. The PMA core has two input clocks, one of which is a low-speed clock on the system side with a value of 156.25 MHz, while the other one resides on the medium side with a value of 3.125 GHz. Based on these two clocks, the system will serialize and de-serialize the data.

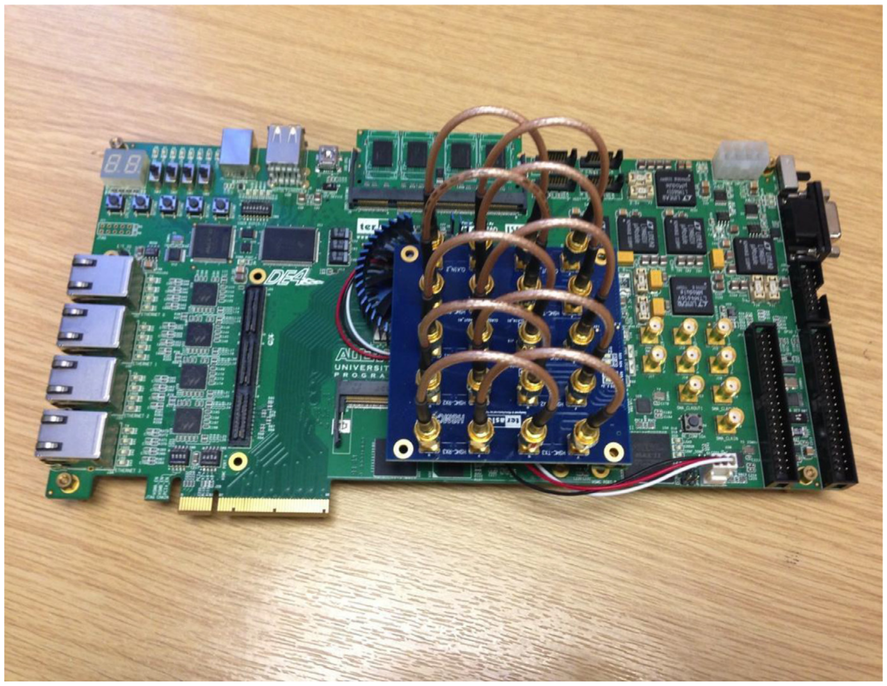
Fig 7. The HSMC's TXs and RXs connecting together through SMA cables and an XTS board plugged into the DE4 board [12, 20].

For the SLMC system the length value of the transceiver packet was changeable depending on the number of repeated pixels inside the original image. As a result, the compression core sent the length of the compressed packet to the transceiver devices, in order to avoid taking any extra bytes when carrying data. This value would be used on the other system side when decompressing the image, in order to identify compressed data from the table. The way this value was sent to the next core depended on the transceiver device and protocol.

3. SLMC system implementation

The SLMC system is a combination of a transceiver system and a compression system. The SLMC has an architecture which divides the system architecture into three layers: application, transport and physical. The compression is part of the application layer, while the physical layer consists of multiple single transceivers connected in parallel. The SLMC system was fully implemented inside the FPGA device, consisting of three layers, each of which included different blocks which were internally connected. The embedded system inside the FPGA interfaced from one side with the application device and from the other side the transceiver physical medium.

The SLMC system is implemented as fully hardware by developing IP blocks, which are created using Quartus-II software and then implemented inside the FPGA as an embedded hardware system. Each IP-block is designed for a specific purpose, and each layer of the SLMC system consists of one or more blocks. These IP blocks are connected together internally using Altera's Qsys logic [19].

The system was tested by programming the FPGA device with the embedded system and then monitoring the FPGA device with the Quartus-II signal-tap analyzer tool. Inside the system a counter was implemented which was used to record the number of cycles between reading the first byte from the source and writing the last byte to the destination memory for the various implementations. The value of the counter was accessible by the signal-tap analyzer tool. The counter was triggered by the same master clock used to synchronize the system. The master clock that used for the compression system, the transceiver and the SLMC systems with the LVS, the GIGE protocols was equal to 125 MHz, while for the transceiver and SLMC based on the Rapid-IO protocol the clock-value was either 156.25 MHz, 195.3125 MHz or 312.5 MHz.

Terasic's DE4 board [12] was used as a testing platform for the SLMC system using LVS as a transceiver protocol. It was tested by exporting the transmitter and receiver signals to banks 2 and 3 on a HSMC extender board (see Fig. 7) and then connecting them together using an external loopback board. However, for the SLMC system based on the GIGE, and the Rapid-I/O protocols, the full system was implemented as an embedded system with internal loopback links. The system ports were connected directly to the HSMC board's bank 1. This bank 1 consisted of eight ports assigned with a Pseudo Current Mode Logic (PCML) voltage, with each port supporting up to 6.5 Gbps, meaning that bank 1 is suitable for different transceiver protocols with different speeds. Both systems were tested in two ways. For the full transceiver system, this was achieved by connecting two transceiver systems together, exporting each one to a separate HSMC board, either on the same board as Fig. 7 shows, or a separate testing board such as two DE4-boards, using an SMA cable with an XTS board, this system supports up to 16 Gbps as a maximum theoretical transceiver speed with the GIGE protocol and up to 40 Gbps with the Rapid-I/O protocol.

4. Test and experimental results

The System implementation is based on logic block (or IP cores) that are written in Verilog HDL using Quartus-II software and then implemented inside the FPGA as an embedded system, while the hardware implementation means the necessary physical elements, such as a board, an FPGA device and transceiver ports. The SLMC systems based on different protocols were tested in two parts by determining the compressed image size, the compression ratio, the actual execution time inside the system, the compression time, the transmission time, and the pixel-rate when applying a sample of medical images. Both the compression time and the transmission time are determined by using two embedded counters, each of which has two triggers and is connected in two different ways in order to find both values. To determine the compression time, the first trigger is triggered when the compression core reads the first pixel from the packet source, and the end-trigger is triggered when finishing forwarding the last compressed pixel to the next core at the transport layer. To determine the transmission time, on the other hand, the start trigger is triggered when the first pixel is read from the transport layer, and the end-trigger is triggered when the final pixel is sent out to

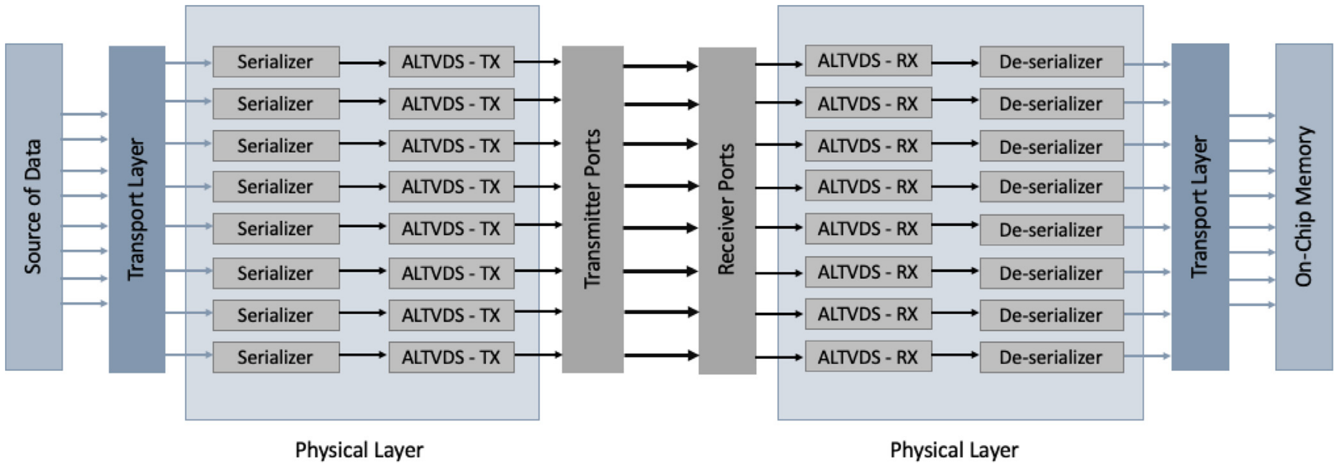

Fig 8. An SLMC-LVS schematic diagram of the transmitter and receiver ports.

Table 1
The compression time, the transmission time, the execution time and the pixel rate when applying a sample of greyscale medical images to the SLMC-LVS system.

The frequency of:	Minimum value	Maximum value	Mean value
Compression-time (ms)	0.00622	0.148	0.139
Transmission-time (ms)	0.00411	0.0696	0.0655
Execution-time (ms)	0.0623	0.15	0.141
Pixel-rate (Giga pixel/*sec*)	0.436	1.05	0.467

Table 2
The compression time, the transmission time, the execution time and the pixel rate when applying a sample of greyscale medical images to the SLMC-GIGE system.

The value of:	Minimum value	Maximum value	Mean value
Compression-time (ms)	0.00622	0.148	0.139
Transmission-time (ms)	0.00488	0.0704	0.0663
Execution-time (ms)	0.0624	0.15	0.141
Pixel-rate (Giga pixel/*sec*)	0.436	1.05	0.467

the physical medium. However, in order to find the total execution time for the SLMC systems, the longer time between the compression time and the transmission time will be the execution time. The compression time is longer than the transmission time, because the compression time will include the time inside the compression core to read, scan and check the input image and then generate the compressed image. The transmission time is the time required to read and add the header file, and then send the data to the physical medium. The transmission time mainly depends on the size of the compressed image and the input frequency of the transceiver devices, which differ depending on the protocols used, for example the LVS and the GIGE run at 125 MHz, while the Rapid-IO runs at 312.5 MHz. With this variety of transceiver protocols, the transmission time can be decreased by increasing the value of the input frequency. The transmission time and the pixel rate of the SLMC systems were compared with the transmission time and the pixel rate of the transceiver system for a similar protocol, when applying a sample of greyscale medical images, in order to assess the benefits of compressing data before transmission, compared to transmitting without compressing.

While the second part runs the whole system in MATLAB by applying around 300 greyscale medical images; based on the equations concluded from the first part, the performances are calculated in MATLAB for a wider sample. The MATLAB calculation is mainly based on reading the image from the computer, finding their histograms, compressing them based on the RLCM in the same way applied in the hardware, and finding the following performances: the compressed image size, the compression ratio (CR), the compression time, the transmission time, the execution time, and the pixel-rate.

At the SLMC-LVS system that shown in Fig. 8. The CR values ranged between 0.9 and 15.9. Table. 1 shows the transmission time, the compression time, the execution time and the pixel rate for the applied sample of images. The compression time values was between 0.0621 ms and 0.148 ms, all of which suited the real-time process, while the average value of the compression time was 0.139 ms. The value of the transmission time varied for the taken images between 0.004 ms and 0.069 ms when 0.065 ms was the average value of the transmission time. The transmission time was very short compared to the compression time because the number of cycles the system required to compress the image was much larger than the number of the required cycles to transmit the compressed image. The value of the execution time for the taken images varied between 0.0623 ms and 0.15 ms when 0.141 ms was the average value of the execution time; it also suited the real-time process. Based on the execution time, the pixel rate (i.e. the number of pixels carried within one second) was measured. The pixel rate values ranged between 0.436 Giga pixel/*sec* and 1.05 Giga pixel/*sec* when 0.467 Giga pixel/*sec* was the average value.

The second model is the SLMC-GIGE, which was tested as an embedded system in the FPGA device. Fig. 9 shows the schematic diagram of the SLMC-GIGE at the transmitter and receiver sides. Both the transmitter and the receiver were exported to bank 1 on the HSMC connector, whether to the same HSMC and connecting them together through a HSMC loopback header, or to two HSMCs and connecting them through an XTS board with SMA cables. Similar to the previous system, the testing case was reading greyscale medical images with a size of 256×256-pixels. The image was divided into eight packets with a size of 8K-pixels; each packet was processed and transferred through an individual transceiver system to another application on the other side.

The compression time, the compressed image size and the CR for the applied medical images were similar to the values at the SLMC-LVS, in order to use the same method to compress data at both cases with the same value for the input frequency. The transmission time, the execution time and the pixel rate for the applied images are shown in Table. 2; these three factors mainly depended on the transceiver protocol. The system execution time, which is the total delay in the system due to compress and transmit data, is equal to the summation of the longest compression time taken at one of the parallel ports in addition to the transmission time for the final sub-packet. Both the transmission and compression time changed according to the CR for each input image; when the CR was high, the execution time was short, and the pixel rate was high. However, when the CR was low, the execution time was

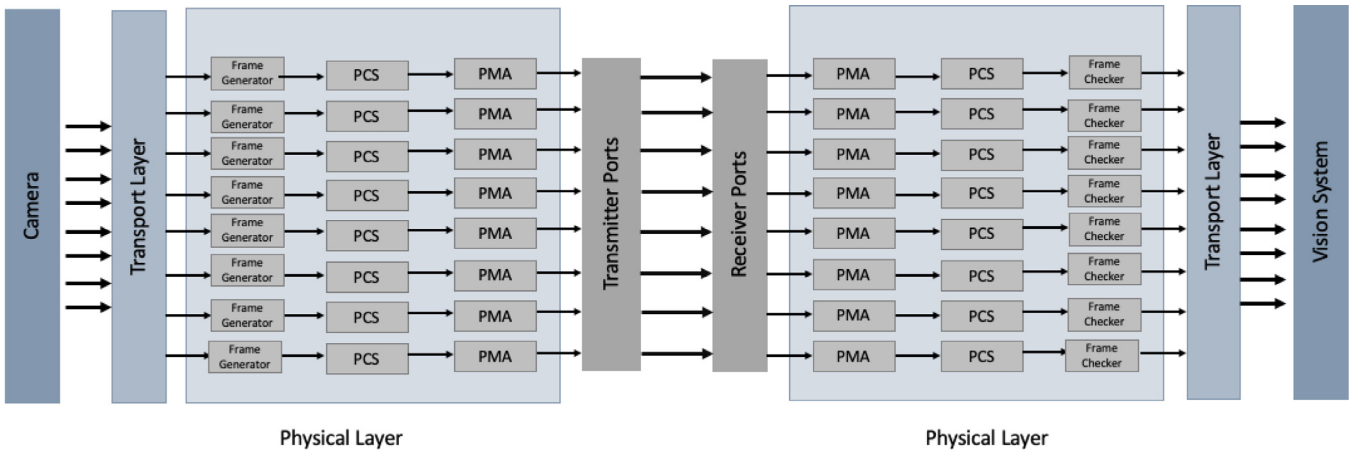

Fig 9. An SLMC-GIGE schematic diagram of the transmitter and receiver ports.

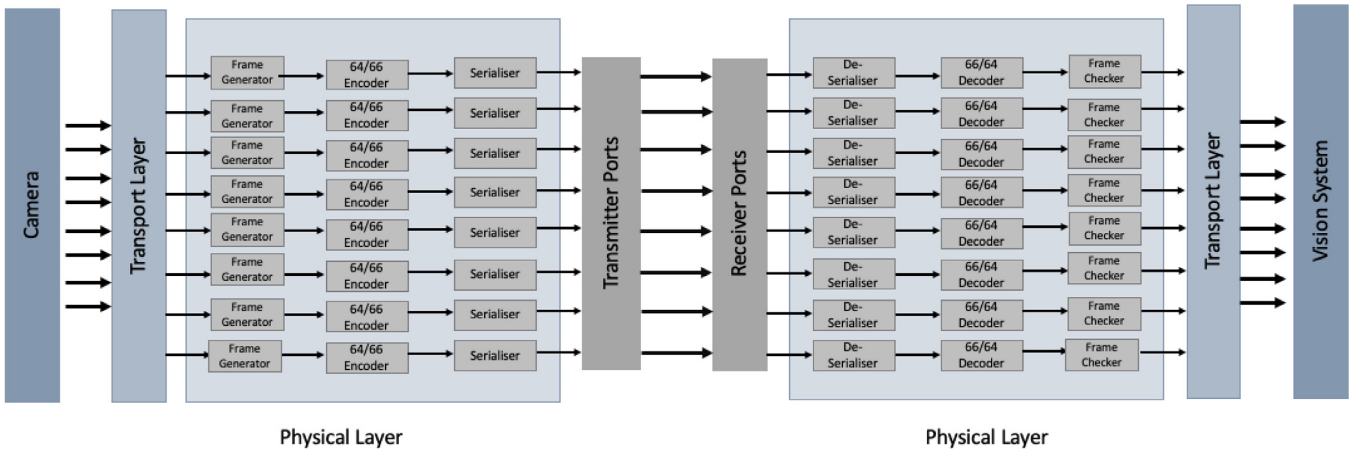

Fig 10. An SLMC-RapidIO schematic diagram of the transmitter and receiver ports.

long, and the pixel rate was low. The compression time values of different images ranged between 0.0622 ms and 0.148 ms, while the average value for the compression time was 0.139 ms. However, when considering the possibility of differences between the measured and the calculated compression time; the compression time values of different images range from 0.054 ms to 0.156 ms. The transmission time varied between 0.0048 ms and 0.07 ms with an average of 0.0663 ms. This time is very short compared to the time it takes to compress the image. Meanwhile, the time required to compress and transmit the whole image, called the execution time, varied between 0.0624 ms and 0.150 ms when the average value was 0.14 ms; all three-time values allowed the image to be compressed and transmitted in real time. The pixel rate of the SLMC-GIGE system varied between 0.436 Gpixel/*sec* and 1.05 Gpixel/*sec* with an average of 0.467 Gpixel/*sec*.

The third model is the SLMC-Rapid-IO that shown in Fig. 10, which was tested by implementing the system as an embedded hardware system inside the FPGA device and exporting both the transmitter and the receiver signals to bank 1 on the HSMC, whether to the same HSMC and connecting them together through a HSMC loopback header, or to two HSMCs on the same board, or two separate boards, and connecting them through an XTS board with SMA cables.

The testing case was reading greyscale medical images with a size of 256 × 256-pixels, which is divided into eight packets with 8K-pixels. Each packet is compressed inside one core and transmitted through a single transceiver system. With each compression core and single transceiver system an embedded counter was used to measure the compression time and the transmission time. For each image, six factors were calculated: the compression time, the transmission time, the execution time, the size of the compressed image, the CR and the pixel rate. The compression time was mainly measured in view of the number of cycles that the system required to compress the image; it was similar to the previous cases with the LVS and the GIGE when the input frequency was 125 MHz. This time can be reduced by increasing the input frequency. The transmission time was variable for each image depending on the protocols used. With the Rapid-IO transceiver protocols, three frequencies were used as input frequencies: 156.25 MHz, 195.3125 MHz and 312.5 MHz. The compressed image size and the CR were similar to all the protocols since these values only depended on the way the image was compressed, inside one core or eight cores, no matter what transceiver protocols were used. The pixel rate also varied according to the CR which was calculated based on the size of the input image that was fixed for all of them, and the execution time was different depending on the CR and the input frequency. The compression time, the transmission time, the execution time, and the pixel rate were measured in MATLAB as shown in Table. 3 when applying a wide range of medical images. All the values were different depending on the input frequency and the CR values; the average value of the compression time was 0.11 ms at 156.25 MHz, while it was 0.0892 ms at 195.3125 MHz and 0.0557 ms at 312.5 MHz. For the transmission time, the average value was 26.8 µs at 156.25 MHz, while it was 21.5 µs at 195.3125 MHz and 13.4 µs

Table 3
The compression time, the transmission time, the execution time and the pixel rate when applying a sample of greyscale medical images to the SLMC-Rapid-IO system with three frequency values (156.25 MHz, 195.3125 MHz and 312.5 MHz).

The value of:	At frequency:	Minimum	Maximum	Mean
Compression-	156.25 MHz	4.98×10^{-5}	3.98×10^{-5}	2.49×10^{-5}
time	195.3125 MHz	1.18×10^{-4}	9.48×10^{-5}	5.92×10^{-5}
(*sec*)	312.5 MHz	1.11×10^{-4}	8.92×10^{-5}	5.57×10^{-5}
Transmission-	156.25 MHz	2.26×10^{-6}	1.18×10^{-6}	1.13×10^{-6}
time	195.3125 MHz	2.85×10^{-5}	2.28×10^{-5}	1.42×10^{-5}
(*sec*)	312.5 MHz	2.68×10^{-5}	2.15×10^{-5}	1.34×10^{-5}
Execution-	156.25 MHz	4.98×10^{-5}	3.99×10^{-5}	2.49×10^{-5}
time	195.3125 MHz	1.19×10^{-4}	9.55×10^{-5}	5.97×10^{-5}
(*sec*)	312.5 MHz	1.12×10^{-4}	8.98×10^{-5}	5.61×10^{-5}
Pixel-	156.25 MHz	5.49×10^{8}	6.86×10^{8}	1.1×10^{9}
rate	195.3125 MHz	1.32×10^{9}	1.64×10^{9}	2.63×10^{9}
(pixel/*sec*)	312.5 MHz	5.88×10^{8}	7.35×10^{8}	1.18×10^{9}

at 312.5 MHz. However, for the execution time, which was the time required to compress and transmit the whole image, the average value was 0.112 ms at 156.25 MHz, while it was 0.0898 ms at 195.3125 MHz and 0.0561 ms at 312.5 MHz, almost equal to the compression time regarding the small time spent to transmit the packet compared to the time required to compress it. All these values met the real-time condition (i.e. compressing and transferring the image within a time equal or shorter than the time required to read a new image). Finally, the average pixel rates were 0.588 Gpixel/s, 0.735 Gpixel/s and 1.18 Gpixel/s when the input frequencies were 156.25 MHz, 195.3125 MHz and 312.5 MHz, respectively. When comparing the SLMC-Rapid-IO pixel rate with the pixel rate at the transceiver system with the same protocol, we found out that the image at all the CR values could increase the number of pixels that could be carried within one second, while at the transceiver system, these values would be fixed because the size of the carried image was fixed as equal to the input image size (256×256-pixel).

In terms of transmission time, which is directly dependent on the compressed image size, and when comparing the range of the transmission time provided by the SLMC system (which was between 4.11 µs and 96 µs) with the transmission time at the system that only transferring data (without compression) through a single line (which was 0.52 ms), the gain in the transmission time will be between 7.5 and 126.8. That means the impact of compressing the image inside parallel cores and carrying the compressed data through parallel lines (SLMC system) is similar to the impact of compressed the same image inside a single core with a higher compression ratio, in this case between 7.5 and 126.8.

5. Conclusion

The SLMC was designed and implemented as a new reference model for the multi-link system to carry images, in real-time, with a high resolution at high transceiver speeds between two applications over physical media. The SLMC is based on a combination of two subsystems for transferring and compression. The transceiver subsystem has been implemented transceiver protocols, such as the LVS, the GIGE and the Rapid-IO, in order to match different applications, physical media and speeds, while the compression system was implemented and used because the main data for the system were an image, in which case the application layer consisted of a compression method that would reduce the size of an image and subsequently increase the system's data rate. The SLMC physical layer consisted of eight single transceiver devices connected in parallel in order to support higher data rates. The scalability at the SLMC system can be achieved in two different ways: at the compression system and at the transceiver system. At the compression system, there are two ways available to decrease the compression time in order to process the image in real-time. First, when using a single core to compress the whole image, the compression time can be decreased by increasing the clock frequency; the maximum value of the input frequency depends on the FPGA device. Secondly, by dividing the image into multiple sub images and using multiple cores to compress the sub images in parallel. Similarly, the transmission-time of the image through eight ports is almost one-eighth of the transmission-time spent when carrying the image through a single line. While at the transceiver system, the scalability was mainly focused on reducing the transmission time that caused increment in the system data rate. On one hand, using parallel lines to carry the data instead of a single line helped to increase the data rate by almost eight times, while on the other hand, using various transceiver protocols that can be worked in different physical media and support various data rates triggered by different input frequency values, viz. 125 MHz, 156.25 MHz, 195.3125 MHz and 312.5 MHz. The number of the transceiver systems connected in parallel is limited by the available number of output pins. The FPGAs have a variable number of transceiver links, only eight were used in our case. However, with other FPGAs when the number of output pins is larger, the system can consist of more cores connected in parallel and directly connect to the input/output ports thus allowing users to scale up the system.

Declaration of Competing Interest

'The authors declare that there is no conflict of interest'.

References

[1] G. Sun, Z. He, in: A Real-Time Multi-Channel Signal Acquisition Card Based On PCI Express Interface, IEEE, Macau, 2009, pp. 20–24.
[2] G. Zhen, Y. Zhang, Y. Ren, in: The Design of PCI Express-Based Multi-Channel LVDS Signal Transfer Card, IEEE, Dalian, 2011, pp. V3.139–V3.141.
[3] Q. Wu, J. Xu, X. Li, K. Jia, in: The Research and Implementation of Interfacing Based On PCI Express, IEEE, Beijing, 2009, pp. 3.116–3.121.
[4] H.-.Y. Huang, L.-.W. Huang, W.-.S. Tseng, C.-.Y. Hsu, in: A 6-Gbit/s SATA Spread--Spectrum Clock Generator Using Two-Stage Delta-Sigma Modulator, IEEE, Newport Beach, CA, 2008, pp. 333–336.
[5] W. Wu, H.b. Su, Q.z. Wu, in: Implementing a Serial ATA Controller Based On FPGA, IEEE, Changsha, 2009, pp. 467–470.
[6] R. Subramani, R. Penneru, G. Selvaraj, B. Radhakrishnan, in: Coverage Metrics for Device Level Validation of SATA and SAS Devices - An Approach, IEEE, Langkawi, 2014, pp. 163–168.
[7] W. Liu, X.-m. Zhao, B.-.J. Hu, X. Zhang, in: A High-Speed Large-Capacity Embedded Storage System Based On SATA, IET, Xi'an, 2013, pp. 1–6.
[8] A. Corporation, Implementing SATA and SAS Protocols in Altera, Altera, Devices, s.l., 2012.
[9] Ri-Kun Liao, Yue-Feng Ji, Hui Li, in: Optimized Design and Implementation of TCP/IP Software Architecture Based on Embedded System, IEEE, Dalian, 2006, pp. 590–594.
[10] Yan Hongwei, Pan Hongxia, in: The Design and Implementation of Network Data Link Layer Based on Embedded TCP/IP Protocol Stack, IEEE, Manila, 2010, pp. 227–230.
[11] S. Kim, C. Park, S. Kim, Y. Chung, in: The Offloading of Socket Information For TCP/IP Offload Engine. International Conference On Advanced Communication Technology, IEEE, 2009, pp. 826–831. 2009. ICACT.
[12] R.B. Mohammed, Novel Scalable and Real-Time Embedded Transceiver System, University of Manchester, 2017 PhD thesis.
[13] Zhong-Zhen Wu, Han-Chiang Chen, in: Design and Implementation of TCP/IP Offload Engine System over Gigabit Ethernet, IEEE, Arlington, 2006, pp. 245–250.
[14] K. Hashimoto, V. Moshnyaga, in: A New Approach For TCP/IP Offload Engine Implementation in Embedded Systems, IEEE, Pacific Grove, 2010, pp. 1249–1253.
[15] E. Chang, C. Wang, C. Liu, K. Chen, C Chen, Virtualization Technology for TCP/IP Offload Engine, IEEE Trans. Cloud Comput. (2014) 117–128.
[16] Yong Ji and Qing-Sheng Hu, 2011. 40Gbps multi-connection TCP/IP Offload Engine. Nanjing, IEEE, pp. 1–5.
[17] Z. Ye, H. Q. and P. Weijun, "Application of High Speed Serial Data Transmission System in Remote Sensing Camera," in MATEC Web of Conferences, 2017.
[18] Rand B. Mohammed, Roelof van Silfhout, Lossless Compression Methods for Real-Time Images, Eurasian Journal of Science and Engineering 6 (2) (2020) 169–188 https://eajse.tiu.edu.iq/index.php/volume-6-issue-2-article-14/, doi:10.23918/eajse.v6i2p169.
[19] TerasicDE4 User Manual -2015, Terasic, 2003 2016.
[20] Rand B. Mohammed, Roelof van Silfhout, Real-time Transceiver System based on Rapid-IO Protocol, Eurasian Journal of Science and Engineering 4 (2) (2018) 39–53 https://eajse.tiu.edu.iq/index.php/volume-4-issue-2-article-4/, doi:10.23918/eajse.v4i2p39.

Dr. Rand Basil Alhashimie is a lecturer in Mechatronics Engineering Department at Tishk International University, Erbil-Iraq. She awarded with a Ph.D. degree in the field of Electronics Engineering from the University of Manchester, United Kingdom in 2017. She obtained an M.Sc. degree in the field of Electronics and Communication Engineering and a B.Sc. degree with First Rank in the field of Electronics Engineering from the University of Mosul, Iraq, in 2010 and 2008 respectively. Her research interests include the development of the embedded systems specifically in the application of Field Programmable Gate Array (FPGA).

He received his PhD from the University of Leiden NL and has worked at the EMBL Hamburg Outstation and at the ESRF in Grenoble. His research interests include high-precision instrumentation, optical characterization methods, and the application of x-rays in structural studies of materials at the atomic scale.