# A Gigabit Ethernet Media Access Controller for TCP/UDP Radar Data Streaming and Visualization

Vukan D. Damnjanović, *Student Member, IEEE*, and Vladimir M. Milovanović, *Senior Member, IEEE*

***Abstract*—A design of a gigabit Ethernet media access controller implemented using Verilog hardware description language is depicted in this paper. The proposed digital hardware module can be utilized for establishing client-server connections over a computer network with a PC, an FPGA-based board or some other separate piece of hardware. It allows users to perform network data transfers using either TCP or UDP communication protocols, in both directions. Data is transmitted to or received from a predefined Internet Protocol address utilizing packets of predefined size, in a format suitable for the corresponding protocol, with a packet header providing the receiving end with the information about the packet itself. The described design is able to achieve network throughput rates that exceed 110 MB/s making it suitable for systems and applications that require high-speed data streaming, such as the system for radar data streaming and PC visualization depicted in the latter part of the paper. Besides that, it can be used in a wide range of applications developed on systems containing boards and devices with the Ethernet 8P8C port as an integral part. The implemented design has been thoroughly tested using a combination of a commercial FPGA development kit and the PC-run Python applications. It was verified and confirmed that the design meets the expectations regarding both the specified functionality and performance.**

***Index Terms*—Gigabit Ethernet MAC, data streaming, UDP and TCP protocols, PC data visualization, Verilog hardware description language.**

## I. INTRODUCTION

During the last couple of decades, there is a growing trend for the amount of different devices used in the systems and applications in practically all the spheres of the IT industry. The same also applies for the systems used for collecting data, such as some sensor-based systems or radar systems. They are becoming more complex, consisting of more devices with more information needed to be carried. Whether it is because of the insufficient available resources, some environmental limitations, inappropriate system topology or something else, the data-collecting devices are often unable to perform the complete cycle of information extraction, processing and utilization relying only on themselves. Therefore, at some point of the cycle, it is necessary that the data is transferred to another device (or devices) so the system can work properly and fulfill its purpose. In most cases, those devices are PCs, due to their abilities and versatility.

Vukan D. Damnjanović is with the School of Electrical Engineering, University of Belgrade, Bulevar kralja Aleksandra 73, 11120 Belgrade, Serbia and also with NOVELIC d.o.o., Veljka Dugoševića 54/B5, 11060 Belgrade, Serbia (e-mail: vukan.damnjanovic@novelic.com).

Vladimir M. Milovanović is with the Department of Electrical Engineering, Faculty of Engineering, University of Kragujevac, Sestre Janjić 6, 34000 Kragujevac, Serbia (e-mail: vlada@kg.ac.rs).

The need for these data transfers is especially emphasized on the distinguished group of systems - real-time systems, where the information acquirement, extraction and/or presentation is performed constantly at relatively high rate [1]. That implies that the data transfers from one platform to the other needs to be performed in the same manner. The amount of data that needs to be transferred and the transfer rate differ from system to system and can vary significantly, which includes some large numbers as well.

In order to achieve the ability of performing these transfers, a vast number of mechanisms have been developed during the years. They can rely on different technologies and all of them have their advantages and disadvantages. Among the most common and popular ones is definitely Ethernet [2].

Ethernet is, technically speaking, a family of wired computer networking technologies, but it usually refers to the most common type of Local Area Networks (LANs) - a connected network of computers (or, to be more precise, devices) in a small area[1]. Devices possessing the Ethernet port and connected through it to the network are able to perform data transfers with other connected devices by following the series of standardized protocols and rules [3]. Ethernet has been developing and improving during the years. It is currently one of the fastest communication technologies.

Computer networks using Ethernet consists of several abstraction layers [4]. In order for the whole mechanism to work correctly, rules for each one of them have to be applied. Following those rules is often managed by some kind of the processing unit, in devices that poses one, but in ones that do not, such as an FPGA-based board, it might be challenging to achieve the flawless operation of the system. The gigabit Ethernet media access controller from this paper's topic is created so that the Ethernet ports can be utilized for preforming data transfers without engaging any kind of processing unit.

This paper, in its first part, gives the quick overview of the data streaming protocols used in the gigabit Ethernet media access controller module, as well as the detailed description of the module's design, along with the description of its implementation using Verilog Hardware Description Language (HDL). In the second part of the paper, obtained testing results and performances are provided with the example of one of the systems which the module was tailored for in the first place.

[1]Computer networks is a large field in network sciences and a lot could be written about it, but the information provided is sufficient for the comprehension of the content of this paper.

## II. A Gigabit Ethernet Media Access Controller and Data Transfer Protocols

A gigabit Ethernet media access controller is a digital component which allows the user to perform data transfers between itself and some other module or device. Basically, the implemented module allows the user to send and receive the data to and from the specified address on the network belonging to some other device. In order for it to work properly, the module requires that the Ethernet port along with the gigabit Ethernet transceiver exist on the device. It is used to set up the transceiver for working in the appropriate mode at the beginning of the application and then to send (or receive) data through the Gigabit Media-independent interface (GMII) to the transceiver [5] and through the port to the network and the rest of the system. Data is fragmented and transferred in packets, where every packet is of the same length and consists of a header and data itself.

### A. Ethernet Abstraction Layers

Currently, two different versions of this module exist: one that supports transfers (receive and transmit) using Transmission Control Protocol (TCP) [6] and another that supports transfers using User Datagram Protocol (UDP) [7]. Those two protocols are parts of the Ethernet transport layer, one of the abstraction layers mentioned in the previous section. This layer provides the end-to-end communication services for applications. This module also secures that the device is working in accordance with two other abstraction layers: link layer, which provides the link to a physical connection of the host, and internet layer, which serves as a bridge between link and transport layer [4]. The fourth and final layer - application layer, can be implemented on the PC or on some other device.

The controller module implements the network link layer by applying the Address Resolution Protocol (ARP) [8]. It sends a message in the appropriate format that provides the physical MAC address of the Ethernet port when another device on the network asks for it.

The internet layer regulates that every message or packet in the network end up at the appropriate destination. The primary protocols for the internet layer are the Internet Protocol (IP). This protocol assigns an IP address to every device on the network, which allows a packet to find its way to the destination. The implemented module utilizes the IP protocol version 4 (IPv4).

The Ethernet transport layer, as mentioned before, is implemented using either TCP or UDP protocol. This layer provides services to the network, such as connection-oriented communication, reliability, flow control etc.

### B. TCP and UDP Protocols

TCP is a more complex protocol than UDP. It is connection-oriented with built-in systems checking for errors and guaranteeing that data will be delivered in the order it was sent. The connection firstly needs to be established, then maintained, and finally terminated, making it a more reliable protocol. All of this, however, requires larger overheads in data packets, which reduces its speed and efficiency. On the other hand, UDP is a simpler, connectionless protocol, faster and more efficient, but it does not provide any recovery options when a data error occurs or when a packet is lost [1].

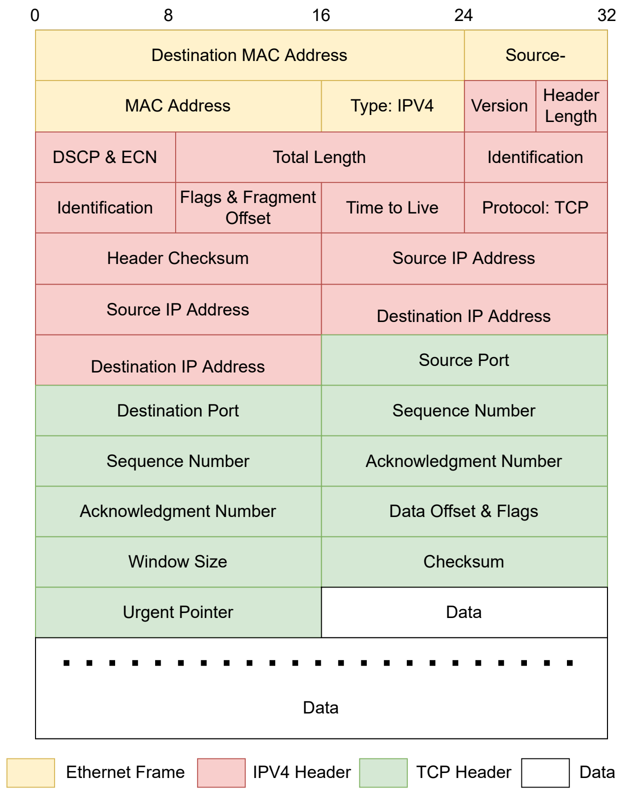

Fig. 1. A structure of data packets of the TCP protocol used in the design.

A structure of data packets of the TCP protocol used in the design is shown in Fig. 1. This type of format allows three network layers to be implemented: the link layer implementation is marked in yellow, the IPv4 header representing the internet layer is marked in red and the TCP header, as a part of the transport layer is marked in green [9]. The link layer carries the information about MAC addresses and the used IP protocol. The IPv4 header has the fields for various information, such as the IP addresses, packet identification number, packet length, header length, used transport layer protocol, the IPv4 header checksum value etc. The TCP header, besides the port numbers, holds the information necessary for establishing, maintaining and terminating the TCP connections, detecting and recovering from errors, flow controlling etc.

The main fields for enabling the TCP to have the connection-oriented communication are sequence number, acknowledgment number, flags and checksum. The sequence number is a 32-bit wide field that carries the number that identifies the first data octet in the packet. The acknowledgment number is also a 32-bit wide field, and it represents a response from the receiving end. It has the value of the next expected sequence number. If the first packet that has arrived had the sequence number of 1 and the N bytes of data have arrived, then the acknowledgment number in the response would be $1 + N$. This mechanism ensures that the order of the received packets is preserved and that there is no packet or data missing. The flag fields have the purpose to indicate that some functionality is being used. There are six flags in total indicating different things: URG - the urgent pointer field is significant, ACK - the acknowledgment number field is significant, PSH - push functionality, RST - reset the connection, SYN - synchronize sequence numbers, FIN - no more data from sender. Basically, to run a TCP connection, only three flags are needed: SYN flag for establishing it, FIN flag for terminating it and ACK flag for acknowledging every received data packet during the connection's life. The checksum field is used for detecting if there is an error in the received data. It has a width of 16 bits, and it is the 16 bit one's complement of the one's complement sum of all 16 bit words in the header and text (only the TCP header and some field of the IPv4 header are included).

In Fig. 2, a diagram that depicts establishing, maintaining and terminating a TCP connection between a client and a server is shown. A client is called a device that initiates the connection and a server is a device that accepts it. Even though it is more usual for the server to send data and for the client to accepts it, it is the other way around in the example shown in Fig. 2. The client initiates the connection by sending a packet with the SYN flag active. The server responds with SYN and ACK flag, which the client acknowledges. At that point, the connection is established. The client then sends N bytes of data in each one of X sent packets, and the server responds for every packet received. Note that it is not necessary for the server to respond to every packet individually. It could also wait for all the packets to arrive and then to acknowledge the reception of them by sending the final acknowledgment number along with the ACK flag. After all the packets are sent, the client expresses the wish to end the communication, which the server accepts and the connection is then terminated.

The UDP data packet structure used in the design is similar to the one used for the TCP protocol. In fact, the only thing that differs is the transport layer protocol header. As mentioned before, the UDP does not provide the possibility of connection-oriented communication, flow control etc. so the UDP header has fewer fields than the TCP header. It only carries the information on the port numbers, packet length and the checksum value. The UDP protocol does not support or require the acknowledgement of the received packets. It straight-forwardly goes to the formation of the following packet, after the previous one has been sent.

## III. The Implementation of the Controller

Previously depicted gigabit Ethernet media access controller have been implemented using Verilog HDL. Its design has been thoroughly tested using standard verification and implementation paths for FPGA design flow. The design is made available [10] by the authors for public use as a free and open-source hardware library.

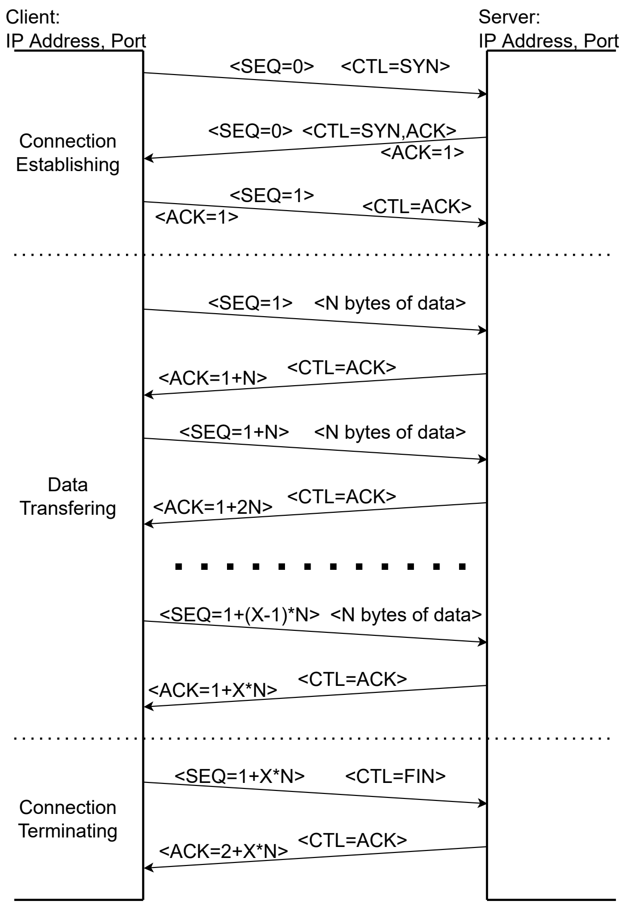

Fig. 2. A diagram that depicts establishing, maintaining and terminating a TCP connection between a client and a server.

The implemented design of the gigabit Ethernet media access controller is relatively complex. It can be divided into several mutually connected submodules, with some of them communicating with the outer world as well, through one of the module interfaces. A block diagram of the module with its submodules and interfaces is depicted in Fig. 3. In this section, descriptions of every individual submodule, implemented interfaces to the outer world and the way the module communicates with other devices will be provided.

### A. Design Interfaces

As it can be seen from Fig. 3, the implemented gigabit Ethernet media access controller has four interfaces. The first one is the AXI Stream interface. The purpose of this interface is to continuously collect data needed to be sent between the devices on the network. The direction of the interface can be both input and output, depending on the fact whether the module is receiving data from some other device, or sending it to the network. The second interface of the implemented module is the AXI4 memory-mapped interface. This interface is used to write values to the memory-mapped configuration registers of the controller, as well as to read status values from it. The next interface is the Reduced

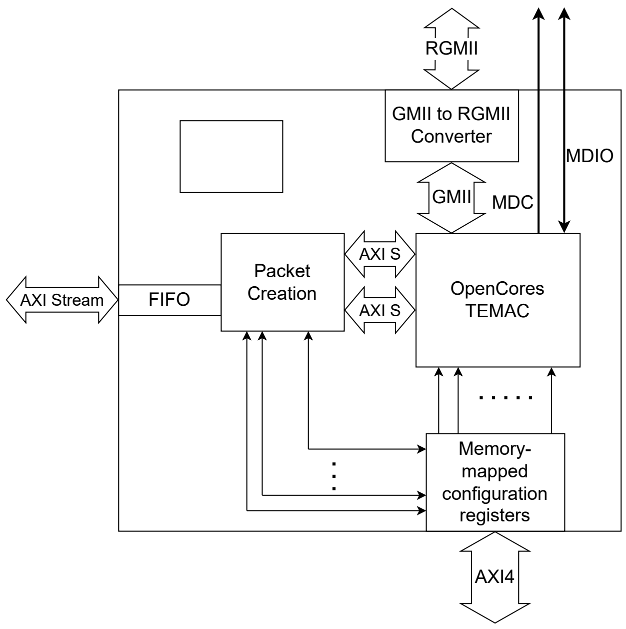

Fig. 3. A block diagram of the implemented module with its submodules and interfaces.

Gigabit Media-independent interface (RGMII), a version of the already mentioned GMII interface. Its role is to communicate with the Ethernet physical layer transceiver that controls the Ethernet port. The last interface is the Serial Management Interface (SMI), also known as Media-Independent Interface Management (MIIM). It is a serial interface used for configuration of the Ethernet physical layer transceivers. In the following paragraphs, the functionality and implementation of every submodule of the controller design will be described.

## *B. Design Clock Domains*

The whole design can be divided into four clock domains. The frequency of the input clock equals 100 MHz, and it drives the clock generator block that creates all other clocks in the module. These four clock domains are the user clock domain, the RGMII physical layer clock domain, the MIIM management interface clock domain and the AXI4 memory-mapped clock domain. The RGMII physical layer clock domain operates at the frequency of 125 MHz, and it is the only mandatory value for all the clock frequencies in the design. The frequency of the MIIM clock domain clock is run-time configurable. Its value depends on the value stored in one of the memory-mapped registers, and it equals the value of the AXI4 memory-mapped clock frequency divided by the register value. The MIIM clock frequency must not exceed 2.5 MHz. The AXI4 memory-mapped clock frequency is the frequency on which the memory-mapped configuration registers and the AXI4 bus operate. It has to be equal to the frequencies of the other memory-mapped devices connected to the bus, and in this version of the design it equals 10 MHz.

## *C. Design Submodules*

The OpenCores Tri-mode Ethernet Media Access Controller (TEMAC) is one of the most important submodules in the design. It is a modified version of an open-source controller downloaded from the OpenCores website [11]. The module has a ”Tri-mode” phrase in its name because, apart from being a gigabit controller, it can also operate as a 100-megabit or 10-megabit. However, in the proposed design, it is utilized solely as a gigabit version. The TEMAC module has several functionalities and five interfaces: an output and an input AXI Stream interfaces, a GMII interface, a MIIM interface and an input interface that provides the module with the memory-mapped register data. Its main function is to convert streaming data-to-be-sent from the input streaming native interface to the output GMII interface and data-to-be-accepted from the input GMII interface to the output streaming native interface. These two native interfaces are converted to the AXI Stream interfaces using the submodule wrapper. For these GMII-to-Stream conversion processes, the module instantiates two dual port block RAMs to serve as asynchronous FIFOs for getting the data from both sides. The finite-state machines (FSMs) control the flow for both directions, from 32-bit streaming data synchronized on the user clock, to 8-bit GMII data synchronized on the GMII clock (or vice versa). The submodule and the exact way it will operate can be set up by reading the input values from the memory-mapped configuration registers. Depending on some of those values, it also generates the MIIM signals for the Ethernet physical layer transceiver configuration.

The GMII to RGMII converter adapts the GMII interface signal to the needed RGMII interface signal. Basically, the RGMII signals are used instead of the GMII signals in order to reduce the number of the occupied output pins. Total number of utilized output pins is halved (12 instead of 24). It is achieved by running half as many data lines at a double speed, time multiplexing the signals and by eliminating non-essential signals. Output pins operate with double data rate (DDR) instead of single data rate (SDR), with the same clocking frequency of 125 MHz. Receiving pins are synchronized to the external clock provided by the Ethernet physical layer transceiver, while transmitting pins are synchronized to the internally generated clock. The clock generating block creates two different 125 MHz clocks with the 90 degrees phase difference, one for the output clock pin itself and the other for the synchronizing of the data and control lines, so that the setup and hold times of the output DDR pins are as large as possible.

The memory-mapped configuration registers module is a submodule whose function is to be accessible from the AXI4 interconnect bus and the rest of the system through the AXI4 memory-mapped interface and to provide the rest of the submodules inside the controller with the written values. It also has a task to inform the OpenCores TEMAC submodule to generate the signals in the MIIM interface. It has numerous registers and here are the most important ones:

- `Physical address` - Value of the address of the Ethernet physical layer transceiver.
- `No preamble` - Indicator whether the transmitting packets will have the preamble to precede them.
- `Clock divider` - Value used to calculate the frequency of the MIIM interface clock.
- `Packet size` - Number of bytes in one data packet, can be up to 1500.
- `PHY data` - Data value to be written to one of the Ethernet transceiver registers.
- `PHY register address` - Register address inside the Ethernet transceiver to which data will be written.
- `PHY write enable` - Indicator that a write operation should be performed through the MIIM interface.

The packet creation submodule is responsible for implementing all the network abstraction layers and protocols in the design. This submodule has several tasks in its jurisdiction. It wraps the data arrived from the streaming interfaces with the appropriate header and calculates all the values for the header fields. It also accepts data packages arrived from the network and checks if they are addressed to this module and creates and sends the response if it is needed. IP addresses, MAC addresses and port numbers for both the client and the server side in this design are hard-coded. For both TCP and UDP versions of the module, this submodule always checks if there is an ARP request sent to the network. If the asked IP address is the one belonging to this module and the ARP format of the message is correct, an ARP response packet is created and sent providing the information about this module's MAC address. Creation of the UDP packets is not too complicated, considering that every field of the header except the identification field is a constant value. The packet creation submodule receives the streaming data and forwards it to the OpenCores TEMAC submodule, except for the occasions when the previous packet has ended and when the header fields are needed to be sent. The end of packet is indicated by sending the active high value for the AXI Stream data last signal.

The situation for the TCP version of the design is a bit more complex. The creation of the header is not as straight-forward as in the UDP version, and the communication between the devices on the network is more complicated. The packet creation submodule calculates the value for several fields for every sent packet, such as the sequence and acknowledgment number, checksum value, flags etc. When the application starts, it sends the synchronization request packet, as depicted in Fig. 2. Then it waits for the response and acknowledges it if the response has the appropriate form and values, and starts creating data packets and streaming data. At the end of the application, it waits for the data acknowledgment message, and then it terminates the connection as described in the previous section.

## IV. Testing Results and Streaming Radar Data Visualization Example

In this section, the design testing flow will be presented. During these tests, the functionality of the design was verified,

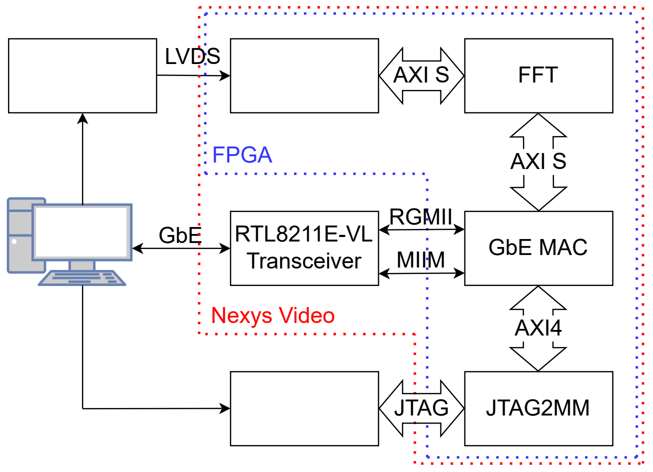

Fig. 4. A simplified block diagram of the complete radar data PC visualization system.

performances and resource utilization were measured, and the design validity is shown as it is used in the example system for radar data visualization. The first step in the design testing flow were the software simulations in the form of the testbench files written in Verilog and VHDL languages.

The next step is the implementation and verification of the design on an FPGA-based development board. A Digilent's Nexys Video board with Xilinx Artix-7 FPGA family is used for it. Nexys Video development board has the Realtek RTL8211E-VL Gigabit Ethernet Transceiver [5] as an integral part of it, making it suitable for the depicted design. For the design testing, some additional features were needed. An open-source JTAG-to-memory-mapped bus master bridge [12] for accessing the memory-mapped register space was used. An alternative for it can be Xilinx's JTAG-to-AXI4 Master module [13]. Write data transactions through AXI4 memory-mapped interface are initiated from the PC using the Python PyFTDI library and the FTDI's cable containing the FT232H chip [14]. Also, a server was run on the PC in order to generate responses for the arrived packets and to send data to the board. For running the server, Socket Python library was used. The arrived packets can be verified by utilizing software for the network packet monitoring, such as Wireshark [15]. Moreover, packets with previously defined data were sent from the board and checked on the PC using Pyhton scripts, therefore proving the correctness of the proposed design. All the examples and testing systems presented in this paper use 1066-byte long packets (52 bytes for the header and 1024 data bytes).

During the hardware implementation testing, data throughput measurements were done. It was proven that the design meets the performance expectations with the maximal data throughput of around 110 MB/s for both TCP and UDP design versions, making it around 90% of theoretically ideal value of 125 MB/s or 1 Gb/s. The resource utilization is moderate, but with less than 10% utilization for all the resource types and with the possibility to reduce it even more.

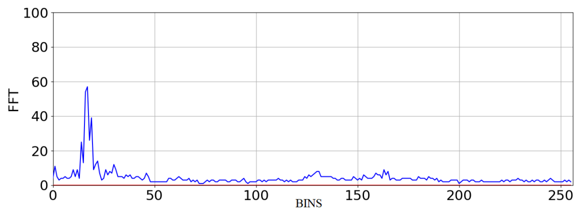

Fig. 5. An example of the radar data plot using Python libraries.

### *A. Streaming Radar Data Visualization Example*

The implemented design of the gigabit Ethernet media access controller, due to the extensive usage of computer networks and its functionality, could find its way into a wide range of different systems. In this subsection, a system for the TCP radar data streaming and PC visualization, one of the mentioned example systems, will be presented.

A simplified block diagram of the radar data visualization system is given in Fig. 4. The system is used to continuously collect data from the radar board, have the incoming streaming data processed (fast Fourier transformation) on the FPGA-based development board, and send it using the gigabit Ethernet to the PC, where the arrived data is plotted. The PC is showing the live display of the distance between the radar board and the detected targets. An example of such a display can be seen in Fig. 5. It should be emphasized that all the processing and data transferring is realized completely in hardware, without involving any kind of processing unit.

For the radar board, Texas Instruments' AWR2243 BOOST [16], along with the MMWAVE-DEVPACK and FMC-ADC-ADAPTER, is chosen. The output is in the form of the Low-Voltage Differential Signaling (LVDS) lines [17] containing radar data, clock and frame clock. Those LVDS lines are connected to the FPGA pins on the Nexys Video board, and they are received and converted to the 32-bit AXI Stream interface. The logic behind this conversion is not relevant for the matter and therefore not elaborated. The AXI Stream data is then driven to the input of the open-source fast Fourier Transformation processor module [18] available to simultaneously perform the processing and stream the output data to the gigabit Ethernet media access controller from the topic of this paper. The TCP packets are created there and sent to the PC through the gigabit Ethernet, where there is a Python script running the server and receiving and plotting data acquired from the Ethernet port using the Python's Matplotlib library. The radar board is previously configured using the MMWAVE Studio software [17] for the PC and the USB interface, as is the gigabit Ethernet MAC module using the JTAG-to-memory-mapped bus master bridge, depicted in the previous section.

## V. Conclusion

In this paper, a design of the gigabit Ethernet media access controller for the UDP and TCP data streaming implemented using Verilog HDL is proposed. This module can be used in a wide range of different systems, due to the nowadays' constant presence of the computer networks in many industrial spheres. One of those systems, or the system for the processed radar data PC visualization to be more precise, is depicted in this paper as well.

The generated instances of the JTAG to memory-mapped bus master bridge were tested and verified by both using software simulations and mapping onto a commercial FPGA development board, proving the correct functionality of the design. The hardware implementation also proved the competitiveness of the design in terms of performances, having the data throughput of over 110 MB/s.

It should be noted that this is only the first version of the design, and there is still a lot of space for improvement and for broadening the functionality of the module. Making it more parameterizable, completely run-time configurable, having better mechanisms to recover from data loss or error etc. are just some of the things that could be and hopefully will be improved in some future versions.

## Acknowledgements

The authors would like to thank NOVELIC d.o.o. for financially and logistically supporting the work on this project.

## References

[1] S. Tibor, P. Dukán, B. Odadžíc, and O. Péter, "Realization of reliable high speed data transfer over udp with continuous storage," in *2010 11th International Symposium on Computational Intelligence and Informatics (CINTI)*, 2010, pp. 307–310.
[2] *The Ethernet*. Digital Equipment Corporation, Intel Corporation, Xerox Corporation, 1982, a Local Area Network, Data Link Layer and Physical Layer Specifications.
[3] V. Cerf and R. Kahn, "A protocol for packet network intercommunication," *IEEE Transactions on Communications*, vol. 22, no. 5, pp. 637–648, 1974.
[4] I. E. T. Force, "Requirements for internet hosts – communication layers," in *RFC*, October 1989.
[5] *Integrated 10/100/1000M Ethernet Transceiver*, version 1.6 ed., Realtek, April 2016, track ID: JATR-3375-16.
[6] "Transmission control protocol," in *RFC*. Information Sciences Institute, University of Southern California, 1981, no. 793.
[7] J. Postel, "User datagram protocol," in *RFC*, 1980, no. 768.
[8] D. C. Plummer, "An ethernet adress resolution protocol," in *RFC*, 1982, no. 826.
[9] W. Zhang, Z. Wei, X. He, P. Qiao, and G. Liang, "The design of high speed image acquisition system over gigabit ethernet," in *2010 IEEE International Conference on Wireless Communications, Networking and Information Security*, 2010, pp. 111–115.
[10] V. D. Damnjanović and V. M. Milovanović, "Gigabit ethernet mac," www.github.com/milovanovic/gbemac, accessed: 2022/04/15.
[11] OpenCores, "Tri-mode ethernet mac," www.opencores.org/projects/ethernet_tri_mode/, accessed: 2022/04/15.
[12] V. D. Damnjanović and V. M. Milovanović, "A chisel generator of jtag to memory-mapped bus master bridge for agile slave peripherals configuration, testing and validation," in *2021 IcETRAN Proceedings*. ETRAN Society, Belgrade, 2021, pp. 239–244.
[13] *JTAG to AXI Master v1.2*, Pg174 ed., Xilinx, February 2021.
[14] *FT232H*, version 2.0 ed., FTDI, document No.: FT000288 Clearance No.: FTDI 199.
[15] U. L. R. Sharpe, E. Warnicke, *Wireshark User's Guide*, (version 3.7) ed.
[16] *AWR2243 Single-Chip 76- to 81-GHz FMCW Transceiver*, Texas Instruments, February 2020.
[17] *DCA1000EVM Data Capture Card*, Texas Instruments, May 2018.
[18] V. M. Milovanović and M. L. Petrović, "A highly parametrizable chisel hcl generator of single-path delay feedback fft processors," in *2019 IEEE 31st International Conference on Microelectronics (MIEL)*, 2019, pp. 247–250.