Igor A. Adamovich, Yuri A. Klimov

An FPGA packet communication protocol

ABSTRACT. When creating computer boards with FPGA or application-specific chips, it is often needed to connect several chips. Existing available buses do not have all the properties required by the authors' task at hand: packet transmission, using a small number of GPIO pins, sufficient bandwidth.

We describe a packet communication protocol that uses GPIO pins and has bandwidth up to 10 MB/s at a frequency of 20 MHz.

Key words and phrases: half-duplex communication, credit-based flow control, data serialization/deserialization, finite state machine, shift register, hardware description language.

2010 *Mathematics Subject Classification:* 68M12; 68M10

Introduction

When creating various computing systems using field programmed gate arrays (FPGA) or specialized chips, a problem arises to connect multiple chips for data transmission.

In some cases, it is possible to use widely used interfaces such as I2C or UART, which, however, have a low data transmission rate. The significant advantages of these interfaces are the use of only two input/output (I/O) lines for data transmission as well as their wide distribution: the hardware implementation of I2C or UART interfaces are present in many microprocessors and microcontrollers. There are free implementations of these interfaces for use in FPGA. Therefore, when it is necessary to connect FPGA and microprocessor or microcontroller, these interfaces are often used.

The work of Yu.A. Klimov was supported by a grant from the Russian Science Foundation, project No. 19-71-30004.

© I. A. ADAMOVICH⁽¹⁾ YU. A. KLIMOV⁽²⁾ 2020

© AILAMAZYAN PROGRAM SYSTEMS INSTITUTE OF RAS⁽¹⁾ 2020

© KELDYSH INSTITUTE OF APPLIED MATHEMATICS OF RAS⁽²⁾ 2020

© PROGRAM SYSTEMS: THEORY AND APPLICATIONS (DESIGN), 2020

If one needs to provide high-speed data transmission, he can use high-speed interfaces that utilize either many general-purpose I/O (GPIO) lines or special differential lines.

The buses with many I/O lines (for example, ISA and PCI buses) were conventional when the data transmission frequency was close to the frequency of chip operation. Now their scope is limited due to the widespread use of differential lines. The main drawback of such buses is the need for tight synchronization of many lines: restrictions on the length of lines and difficulty of routing a board with many lines limit the frequency of data transmission. Currently, differential lines (such as PCI-Express, Ethernet, or USB) provide very high speed (from 1 GB/s and higher) and data transmission over a long distance. Differential ports and, in many cases, hardware blocks implement widespread packet interfaces embedded in many FPGA. For example, for working with an Ethernet network, Intel Corp. provides a Triple-Speed Ethernet IP Core¹ block for its FPGAs, and Xilinx Inc. — Tri-Mode Ethernet MAC² block. Third-party blocks are also available for working with the Ethernet network at the UDP layer, for example, the e7 UDP / IP³ block from e-trees.Japan Inc. This block is used in [1] to implement the CoAP protocol for the internet of things. In some cases, researchers develop their packet protocols based on embedded hardware blocks, such as in [2].

In general, their use is justified for a connection of several FPGAs with a high data transmission rate or a connection of the FPGA to a processor that supports a similar interface. Usually, 1–4 Ethernet or PCI-Express blocks embedded in FPGA allows connecting only 1–4 devices to one chip. In the case of specialized chip development, the use of differential lines increases the cost of the chip (licensing of the corresponding hardware blocks is required) and the complexity of its development. Therefore, it may not be suitable in some cases.

Between these “extremes” there are several interfaces (for example, SPI/QSPI and LPC) that use a small number (4–6) of general-purpose input/output (GPIO) lines and operate at frequencies of several tens of MHz (in some cases a hundred MHz). However, the LPC bus is currently rarely used. SPI, in its basic version, has a relatively low speed. The

¹Triple-Speed Ethernet IP Core, <https://www.intel.com/content/dam/www/program/design/us/en/documents/low-pin-count-interface-specification.pdf>

²Tri-Mode Ethernet Media Access Controller, <https://www.xilinx.com/products/intellectual-property/temac.html>

³e7 UDP / IP, <https://e-trees.jp/e7-udp-ip/>

higher-performance version (Quad SPI, QSPI) is a specialized solution (mainly for flash memory chips) that carry a load of compatibility. It makes no sense to adapt such versions to our needs.

We need to integrate the interface with other components of the project and to minimize overhead when transmitting large packets. After the analysis of the existing interfaces, we decided to develop a new interface for data transmission between FPGAs to match the three following requirements:

- symmetry* — both sides can transmit data independently;
- packet transmission support* — up to 128 bytes in the used version;
- small number of GPIO* — 6 of general-purpose input-output lines in the used version.

This article uses the following structure: Section 1 describes the requirements for the interface that arose in the system developed by the authors. Section 2 provides an overview of existing known interfaces based on the requirements. Section 3 describes the designed interface, its physical and logical implementations, and testing results. The conclusion summarize the development results.

1. Interface requirements

Different tasks to be solved, as well as different physical limitations of a computing system, can put forward different requirements for the interfaces used. We formulate the requirements for the interface that were put forward by the computer system developed by the authors.

The main physical limitations are the available number of lines per port and the required data transmission rate. If it is permissible to use a large number of lines, it is possible to use wide buses with a low clock frequency. If it is necessary to provide a high data transmission rate with a small number of lines, then high-frequency differential lines are required. The specific choice of the physical transmission layer also depends on the supported capabilities of the used chips.

The authors' developed computing system should consist of several FPGAs: one control FPGA and several (up to 20) computing FPGAs connected to the control FPGA. The control FPGA belongs to the Xilinx Zynq 7000⁴ FPGA family, and computing FPGA comes from Xilinx

⁴Xilinx Zynq-7000 SoC, <https://www.xilinx.com/products/silicon-devices/soc/zynq-7000.html>

Artix-7 FPGAs⁵ FPGAs. The main difference between these families, which determined the system architecture, is that the Xilinx Zynq 7000 additionally contains ARM cores and peripherals blocks (for example, DDR, Ethernet, and USB controllers) that allow running universal OS Linux on such chip.

The Xilinx Zynq 7020 FPGA has 200 I/O pins, but some pins serve peripherals. As a result, no more than eight input/output pins are available per one port.

In the developed system, the data transmission rate between FPGAs can be about 10 MB/s (total in both directions). It may increase up to 30–40 MB/s in the future. So a frequency of about 20 MHz (and at 60–80 MHz in the future) is needed for data transmission, which is a suitable frequency for general-purpose input-output (GPIO) lines, so differential lines are not required. In the future, if necessary, the transmission frequency can be increased up to 66–100 MHz.

The main logical requirement arising from the specifics of the task is to support symmetric transmission of packets of various sizes (up to 128 bytes).

As will be shown in the next section, many existing interfaces assume a separation of receiver and transmitter on the master and slave: the master can send requests (containing address and data), and the slave executes the received requests. In general, the next request cannot be sent before the previous one is processed, and the slave usually cannot initiate data transmission (or an additional line is required to send interrupts).

In the developed system, computing FPGAs should contain many computing blocks, the interactions with which should occur independently and in a non-blocking manner. Consider packet transmission, in which all the necessary information is packed into packets and the transmission protocol sends packets without understanding what kind of data they contain.

Using packet transmission allows implementing such essential requirements as symmetry and non-blocking. The symmetry of the protocol allows transmitting data in any direction without any particular activity on the other side.

Non-blocking is also important: the ability to temporarily stop the transmission in one direction (if the recipient is not ready to receive

⁵Xilinx Artix-7 FPGA, <https://www.xilinx.com/products/silicon-devices/fpga/artix-7.html>

FIGURE 1. UART interface schema

new data) while maintaining the ability to transmit data in the opposite direction.

Last but not least, the interface implementation must be compact enough to fit 20 ports in a small FPGA.

2. Overview of existing interfaces

This section will cover existing interfaces and, first of all, their interaction protocols, as well as other features if necessary.

2.1. I2C

I2C (Inter-Integrated Circuit)⁶ bus is an asymmetric serial bus developed by Philips Semiconductors. It uses two bi-directional communication lines: a clock signal line (‘SLC’) and a serial bi-directional data line (‘SDA’). The bus can operate at a frequency of 100-400 kHz, depending on the supported modification, which leads to the very low bandwidth of up to 50 kB/s. Due to the low bandwidth, this bus is unusable in the developed system.

Due to the relative simplicity, low operating frequency, and the small number of lines, many sensors come with I2C bus support. Therefore, most microprocessors and microcontrollers have an embedded I2C bus controller, which has led to the widespread use of the I2C bus to monitor and control various chips. For example, paper [3] discusses the I2C bus for FPGA interaction with MEMS motion sensor.

2.2. UART

UART (Universal Asynchronous Receiver/Transmitter)⁷ is an interface for communication of digital devices, transmitting data in serial form. Two lines are essential for communication: a transmission line and a reception line (Figure 1). It is important to note that there is no clock line, so the transmission frequency is limited to 100 kHz – 1 MHz.

⁶Inter-integrated circuit (I2C), [URL https://en.wikipedia.org/wiki/I2C](https://en.wikipedia.org/wiki/I2C)

⁷Universal asynchronous receiver-transmitter (UART), [URL https://en.wikipedia.org/wiki/Universal_asynchronous_receiver-transmitter](https://en.wikipedia.org/wiki/Universal_asynchronous_receiver-transmitter)

FIGURE 2. Timing diagram of a single word transmission in UART protocol

We describe the basic cycle for transmitting a word (Figure 2). A standard word takes eight bits, but the interface is customizable, and therefore another word size is possible. In the idle state, when there is no transmission, the logical ‘1’ is held on the line. The beginning of the word is preceded by the starting bit ‘0’, so the UART receiver waits for the line to pass from level ‘1’ to level ‘0’, from which the word reception is counted. After the start bit, 8 data bits are transmitted, and after it stopping bit ‘1’ is transmitted.

UART usually connects various sensors to microprocessors and microcontrollers when I2C bandwidth is insufficient or bidirectional interaction is required. In [4], UART interface connects several sensors that measure power consumption. Each sensor has two UART connectors, which allows them to connect a chain. Unfortunately, in this case, the time of receiving data from all sensors will linearly depend on the length of the chain. This work implements several UART interfaces in the FPGA: one for each sensor, as well as an algorithm for parallel polling of all sensors. The result significantly reduces the time of polling sensors.

Due to insufficient bandwidth, UART is unusable in the developed system. However, we note a significant feature of the UART interface: it does not transmit start signal over a separate line (as in some other interfaces described below), but indicate it by a change in the signal level on the data line. Such behavior reduces the number of data lines required, although it increases the time required for data transmission.

2.3. SPI

SPI (Serial Peripheral Interface)⁸ is a synchronous transmission standard, which provides simple and inexpensive connection of microcontrollers and peripherals. In contrast to UART, SPI is a synchronous interface: a clock line generated by the master is used to synchronize the master (microprocessor or microcontroller) with the slave (peripherals). As a result, the transmission frequency can reach 33–66 MHz.

Multiple slave devices can be connected to one master device. In this case, the master will have several additional output lines of the type “chip

⁸Serial Peripheral Interface (SPI), https://en.wikipedia.org/wiki/Serial_Peripheral_Interface

FIGURE 3. SPI interface schema

FIGURE 4. Timing diagram of packet exchange between master and slave in SPI protocol

select”. Peripherals that are not selected by this signal do not participate in SPI exchange at this time (Figure 3).

Transmission in SPI uses words (Figure 4). The word size is usually 8 bits. The master device initiates the transmission by selecting the slave using the “chip select” line. After that, both the master and the slave send word bits on the transmission lines, at the same time they receive the word bits sent by the other side on the reception lines. Thus, an exchange of words occurs: both the master and the slave send and receive one word in a single transmission cycle.

SPI interface is quite common, SPI controllers are embedded into various microprocessors and microcontrollers. Various devices requiring a higher data transmission rate than I2C have the SPI interface. The paper [5] considers the possibility of controlling the digital-to-analog converter with FPGA via the SPI interface.

Paper [6] developed a block for connecting the MicroBlaze microprocessor to devices with an SPI interface. To connect the developed block to the microprocessor, the AXI-Lite⁹ bus interface developed by ARM Holding is used: the MicroBlaze microprocessor sends requests via the AXI-Lite bus

⁹AMBA AXI and ACE Protocol Specification. AXI3, AXI4, and AXI4-Lite, ACE and ACE-Lite, [URL https://static.docs.arm.com/ihi0022/e/IHI0022E_amba_axi_and_ace_protocol_spec.pdf](https://static.docs.arm.com/ihi0022/e/IHI0022E_amba_axi_and_ace_protocol_spec.pdf)

to the block that transmits requests to the external device via the SPI interface. The AXI-Lite bus often combines various blocks in the FPGA. A similar block was developed in the work [7]. Unlike the previous work, it uses APB¹⁰ bus interface, also developed by ARM Holding. The operating conditions of these blocks are significantly different from those required: the AXI-Lite and APB bus interfaces are used, not the packet interface.

QSPI interface (Quad SPI) is a modification of the SPI interface. A significant difference is that the transmission is carried out on four data lines, not on two data lines. Besides, all four lines can provide one-way transmission. This way provides the quadruple one-way transmission rate concerning SPI. QSPI is usually used to connect flash memory chips.

A similar interface usually connects SD-Card flash-memory. SD flash-memory cards can support several communication protocols: standard SPI, 1- or 4-bit SD mode, and the use of differential pairs in UHS-II generation SD-cards. The paper [8] describes a developed block for connecting SD-cards to the APB bus. The developed block can operate at a frequency of up to 281 MHz, and at a frequency of 100 MHz it provides a theoretical data transmission rate of up to 50 MB/s. When using various SD memory cards, the actual data transmission rate was up to 43 MB/s.

Although the data transmission rate is sufficient for our task, the asymmetry incorporated in these protocols (microprocessor — memory) does not allow their use in our conditions.

We emphasize that an important difference between SPI and UART is that a separate clock signal line is used (from master to slave). This line allows SPI to operate at frequencies from 1 MHz to 50–66 MHz and even higher. As a result, SPI and QSPI provide close to the required transmission rate. However, they do not provide the symmetric packet protocol which we need.

2.4. LPC

LPC¹¹ (Low Pin Count) bus is a synchronous bus developed by Intel to connect devices that do not require high bandwidth, such as boot ROM, serial and parallel ports to the system. The bus arose as a replacement for the ISA bus for devices that do not require PCI bus.

¹⁰AMBA Specification (Rev 2.0), https://static.docs.arm.com/ih0011/a/IHI0011A_AMBA_SPEC.pdf

¹¹Intel Low Pin Count (LPC) Interface Specification, <https://www.intel.com/content/dam/www/program/design/us/en/documents/low-pin-count-interface-specification.pdf>

FIGURE 5. LPC interface schema

FIGURE 6. Timing diagram of read request from master to slave in LPC protocol

The specification for the interface of this bus defines six mandatory lines required for two-way transmission (Figure 5): a clock signal, a reset signal, a transmission start signal (frame), and four data transmission lines. There are six additional lines (which may not be available in various implementations) that organize DMA (direct memory access), transmitting interrupts from slave devices, power management, and more.

A 33 MHz clock signal generated by the master synchronizes the master and slaves. Unlike UART and SPI, but similar to QSPI, the LPC bus uses four two-way lines for data transmission.

Each transmission (frame) in the LPC bus is sending a request to the master and receiving a response from the slave (Figure 6). The transmission starts by arising a particular frame start signal. Then, through the data lines, the master transmits a request containing the type of request, address, and data (the full format of requests and responses follows the specification of LPC). After that, the master passes control over the data lines; slave sends a response to the request along the data lines.

Essential for us features of LPC interface are the use of a separate clock line and the use of two-way data lines, which provides the necessary transmission rate. However, the rigid orientation to the asymmetric master-slave separation of devices does not allow the use of LPC in the developed computing system.

FIGURE 7. AXI-Lite based interface schema

2.5. AXI-Lite based interface

Interface AXI-Lite is designed to connect blocks inside microprocessor and FPGA. It has 146 lines, which allows a compact description of the blocks that use it. However, this number of lines does not allow the use of AXI-Lite for communication between chips.

In [9], the author suggests using AXI-Lite to connect two FPGAs, but to reduce the number of required lines, he suggests using serialization/deserialization to transmit a wide range of data over a small number of lines. As a result, the author uses 19 lines instead of 146: nine lines for transmitting data in one direction, nine lines for transmitting in the opposite direction, and one line for transmitting a clock signal (Figure 7).

Separate lines for transmission from the master to the slave and in the opposite direction allows transmission data in both directions simultaneously. A cost is a large number of lines. Our computing system avoids the reset signal and the transmission of a clock signal from the slave to the master. That does not affect the applicability of this approach.

The serialization/deserialization blocks base on the classic shift register and therefore use no particular memory blocks. As a result, the blocks for the master and for the slave devices occupy a small part of the FPGA: 278 LUT¹² and 511 registers in the master, and 62 LUT and 167 registers in the slave.

It is interesting to note that in this interface between FPGAs, there are neither explicit nor specially encoded signals about readiness to receive data or about receiving data. The AXI-Lite interface allows only one transaction between the master and the slave can occur at one time. Therefore, no new data will be transmitted until the current transaction is completed. As a result, it is sufficient to have a small number of registers for storing the current transaction data, which can also be used as shift registers.

¹²LUT — LookUp Table: an elementary block inside the FPGA that implements a logical function (usually from 4–6 arguments) based on the table.

FIGURE 8. Tlink interface schema

Using the AXI-Lite bus and 19 lines between FPGAs do not allow us to use this implementation in the developed computing system.

3. Developed Tlink Interface

3.1. Tlink Interface

Based on the analysis of existing solutions, the authors developed and implemented the Tlink interface in order to optimally connect the control FPGA to some computing FPGAs for the created computing system.

Before describing the actual Tlink interface between two FPGAs, let us describe the user interface by which the other block inside the FPGA (starting now referred to as the “user block”) will interact with the Tlink block. The user interface consists of two sets of lines (one in each direction). Each set consists of data lines (‘data’) and standard handshake signals (‘valid’ and ‘ready’) (Figure 8). This interface is similar to the widely used AXI-Stream¹³ interface, which connects blocks inside the FPGA.

Packets are transmitted along the ‘data’ lines, similarly, for example, to the AXI-Stream interface. Unlike other protocols, the packet start (‘sop’ — start of the packet, the beginning of the packet) and the packet end (‘eop’ — end of the packet, or ‘last’ — the last flit of the packet) signals are not transmitted but are restored based on knowledge of the packet size. Handshake signals are also standard: data on the ‘data’ lines can be transmitted on this clock cycle if both the ‘valid’ and ‘ready’ are equal to ‘1’ on this clock cycle.

¹³AMBA 4 AXI4-Stream Protocol. Specification, [URL https://static.docs.arm.com/ihi0051/a/IHI0051A_amba4_axi4_stream_v1_0_protocol_spec.pdf](https://static.docs.arm.com/ihi0051/a/IHI0051A_amba4_axi4_stream_v1_0_protocol_spec.pdf)

Let us note the essential features of using this user interface that our implementation of the Tlink interface relies on:

- If the user block is ready to receive the packet, it should set the ‘ready’ signal without waiting for the ‘valid’ signal that the packet flit is valid, and it should keep the ‘ready’ signal until the packet is completely transmitted from the Tlink to the user block.
- If the user block is ready to transmit the packet, it should set the ‘valid’ signal, without waiting for the ‘ready’ signal that the Tlink block is ready to receive the packet, and it should keep the ‘valid’ signal until the packet is completely transmitted from the user block to the Tlink.

These rules avoid either buffering and using auxiliary buffers inside Tlink. If it is necessary to bypass these restrictions or to transmit data between blocks that use different frequencies, the user block can use standard intermediate FIFO buffers (first in, first out — queue).

The Tlink interface has six lines: a clock signal (similar to SPI and LPC), a reset signal (similar to LPC), and four two-way data lines (similar to QSPI and LPC) (Figure 8).

To synchronize implementations of the Tlink interface in devices, the master transmits the clock signal to the slave. This feature allows Tlink to transmit data without errors at frequencies up to 66-100 MHz.

The reset signal is transmitted from the master to the slave and is used to reset the slave to its initial state (reset all internal registers).

The four two-way lines provide half-duplex transmission of packets alternately from the master to the slave and in the opposite direction. It is assumed that when developing a board or FPGA program, these lines are pulled up to ‘1’, as in other similar interfaces. This assumption provides a fixed level of ‘1’ on the lines at times when no device drives the lines.

The master and slave devices differ in the Tlink interface exactly in three directions of the transmission for: the clock signal, the reset signal, and the first packet after the reset. Further, when transmitting packets, the devices are symmetrical.

At the line level, the Tlink is close to LPC interface discussed above. However, it does not use a separate start packet transmission line, which allows Tlink to reduce the number of lines used slightly.

Data transmission is carried out in packets from 4 bytes to 128 bytes long. Each packet contains a header (4 bytes) and a packet body (from 0 bytes to 124 bytes).

FIGURE 9. Timing diagram of transmission of one packet in Tlink protocol

The following fields in the packet header are essential for the Tlink protocol: the size of the packet (the size of this field is 5 bits) and the credit information (1 bit), see below.

Packets are transmitted strictly in turn (Figure 9).

First, the master sends the packet, and the slave receives it. Then vice versa: the slave sends the packet, and the master receives it. Then this cycle is repeated indefinitely.

The transition of all data lines from level ‘1’ to level ‘0’ (similar to UART) identifies the beginning of the packet. This rule allows Tlink to avoid using a separate line to indicate the start of the packet. The size of the packet specified in the header identifies the end of the packet.

Changeable parameters of the Tlink implementation allows configuring the Tlink interface for the requirements of various systems. For example, you can specify the number of lines for data transmission, the width of the packet size field and, accordingly, the supported packet lengths, and the header size.

Next, let us look at the implementation of the Tlink interface and the protocol used in more detail.

3.2. Tlink interface implementation

The implementation of the Tlink interface consists of three parts:

- the core of the Tlink interface that implements packet transmission, data lines controlling, initial synchronization, and recovery from incorrect states;
- service layer of the Tlink interface that provides credit information and service packages;
- “observer” of the Tlink interface that provides statistics about the interface operation.

3.3. Tlink core

The core of the Tlink interface must be capable of transmitting and receiving packets. The order of sending and receiving packets is rigidly fixed and described by a finite state machine that has the following states:

- (1) Initialization.
- (2) Waiting for a packet to be sent (WAIT_SERV).
- (3) Sending a packet (SEND_PKT).
- (4) 1st stage of transfer of control over data lines (TAR1¹⁴).
- (5) 2nd stage of transfer of control over data lines (TAR2).
- (6) Waiting for a packet to be received (WAIT_LINK).
- (7) Receiving a packet (RECV_PKT).
- (8) 1st stage of transfer of control over data lines (TAR3).
- (9) 2nd stage of transfer of control over data lines (TAR4).
- (10) Error recovery.

We emphasize that the transmission is asymmetric at the core layer: there are master and slave devices. The differences between the master and the slave are as follows: the master generates a clock signal and a reset signal; both participants go into different states from the initialization state.

After resetting, the master goes from the initialization state into the WAIT_SERV state, but the slave goes into the WAIT_LINK state. The states allow the master to start the transmission and to send the first packet, while the slave receives this first packet. Note that this packet may contain no data and be a service packet.

At the waiting for a packet to be sent state (WAIT_SERV), the level ‘1’ is issued on all four data lines. When the packet arrived at the core input, the level ‘0’ is issued on the data line, and the kernel goes into the sending a packet state (SEND_PKT). When data lines go from ‘1’ to ‘0’, the partner understands that the packet transmission will start at the next clock cycle.

At the sensing a packet state (SEND_PKT), the packet is sequentially transmitted over data lines using a serializer based on a shift register (similar to the AXI-Lite-based interface). The packet header contains information about the packet size, based on which the Tlink core determines when to stop sending the packet.

After the packet is fully sent, the Tlink core transfers control over the data lines to the partner by sequentially transitioning to the TAR1 and TAR2 states. At the TAR1 state, the Tlink core outputs the level ‘1’ on the data lines, and at the TAR2 state, it stops issuing signals on the lines, putting the outputs in a high impedance state. Because the links are pulled up to ‘1’, the level on the lines remains constant and equal to ‘1’. In each of

¹⁴TAR — Turn-Around, deployment of a data line: transfer of control over a data line from one device to another.

the TAR1 and TAR2 states, the core has exactly one clock cycle. After the TAR2 state, the control over data lines goes to the partner, and the kernel goes into the waiting for a packet to be received state (WAIT_LINK).

At the waiting for a packet to be received state (WAIT_LINK), the core reads values on the data lines. As soon as the transition from level '1' to level '0' is detected on all data lines (this means that a packet will be transmitted along the data lines at next clock cycle), the core goes into the receiving a packet state (RECV_PKT).

At receiving a packet state (RECV_PKT), the core sequentially receives the packet using a deserializer based on the shift register. The packet header contains information about the packet size, based on which the Tlink core determines when to stop receiving the packet.

After receiving the packet, the Tlink core takes control over data lines by sequentially transitioning to the TAR3 and TAR4 states. In each of the TAR3 and TAR4 states, the core has exactly one clock cycle. After the TAR4 state, the control of data lines goes to the given core. The core goes into the waiting for a packet to be sent state (WAIT_SERV).

Using the TAR1, TAR2, TAR3, and TAR4 states allow smooth transfer control over data lines from one device to another and eliminate the situation when both devices simultaneously transmit data, which can lead to failure of one of the devices due to high currents on the lines.

3.4. Tlink service layer

The main task of the service layer of the Tlink interface is to ensure symmetric packet transmission, as well as packet transmission only when the receiving device is ready to process a new packet.

This task requires to transmit additional service information via the Tlink interface: information about the readiness of each device to accept data. If the receiving device is not ready to accept data, then it is necessary to transfer control to another device without transmitting the data. For example, if the auxiliary buffers are full and can not accommodate more data, it should be possible to ask the partner not to send more data temporarily.

The special service package for control transfer consists only of a header (without data). As a result, all packets consisting only of the header are processed by the service layer and are not transmitted to the user block. On the one hand side, this prevents the user from sending such small packets by packing all the necessary data into a header. On the other hand side, our system does not use such small packages.

A single “credit” bit in the packet header transmits information about the state of the receiving part of the Tlink interface. It is present in all packages, both service and regular. If the recipient receives a packet with the “credit” bit set, this means that it has the permission to send a data packet in response, and this packet will be successfully received and processed. If the received packet has unset “credit” bit, then the recipient does not have the permission to send a data packet but must send a service packet for control transfer.

The operation of the Tlink service layer is rigidly fixed and based on a finite state machine. The service layer state machine has four states:

- (1) Idle (IDLE).
- (2) Sending a service packet (SEND_META).
- (3) Sending a regular packet (SEND_PKT).
- (4) Receiving a (regular or service) packet (RCV).

One of the main functions of the service part is the following organization of symmetrical data transmission.

After the reset, the master goes into the sending a service packet state (SEND_META), but the slave goes into the receiving a packet state (RCV).

After sending a (regular or service) packet, the service layer goes into the packet receiving state (RCV).

As soon as an incoming packet is received (RCV state), the service layer goes into one of the packet sending state. If the other side of Tlink is ready to receive data (a packet with the “credit” bit set has been received) and there is a packet with data to send, the service layer goes into the sending a regular packet state (SEND_PKT). Otherwise, the service layer goes into the sending a service packet state (SEND_META). In both cases, the “credit” bit is correctly set in the sending packet.

Thus, the packet transmission is symmetrical:

- Both sides of the Tlink send packets in turn, and as soon as their turn comes.
- If the user block has a data packet, and the partner is ready to accept it, the Tlink (the master or the slave) sends this packet. Otherwise, a service packet is transmitted.
- Since a packet (regular or service) will be sent in the opposite direction immediately after receiving the incoming packet, the Tlink will never be blocked.

A user block that uses the Tlink block feels packet transmission to be symmetrical and non-blocking.

3.5. Tlink “Observer”

The third part of the implementation of the Tlink interface is the “observer” block. The “observer” collects statistics on the number of sent and received data packets and service packets. Statistics are collected for the number of clock cycles specified in the block parameters and then issued to the user.

Collecting statistics allows judge how much the Tlink interface is loaded, whether it is a bottleneck in the computing system.

3.6. Comparison

Comparing Tlink with UART, we can highlight the main advantage of UART: it can work at a distance of up to 15 meters. This is achieved due to the low frequency. In the context of the task, such reliability of the data transmission is not required, it is enough to organize the transmission within a single board. The disadvantage of UART is the lack of credit mechanisms and packet transmission; they will have to be implemented on top of the UART protocol. In addition, UART has only one transmission line in each direction, while Tlink has four data lines. Also, UART does not use a separate line for the clock signal: on the receiving side, the clock signal must be adjusted to the received data, which makes it difficult for a high transmission rate.

Compared to SPI, the Tlink interface has fewer “extra” lines since SPI uses an additional device selection line that actively involved in the data transmission. There is no such line in the Tlink, so it uses lines more sparingly. In addition, SPI, like UART, transmits data in two directions at once, which reduces its bandwidth when transmitting a large data stream only in one direction. Note that this problem is not present in QSPI — all four data lines are used in one-way transmission. It is worth noting that SPI and QSPI are asymmetric interfaces — the master initiates requests to slaves. Other disadvantages of these two interfaces are the small and fixed packet size and the lack of credit information support.

Comparing the LPC interface with the Tlink, it is worth noting that LPC uses several more lines. For example, a separate line is used for the transmission start signal, whereas in Tlink, the start of transmission is indicated by the signal level on the data lines. Besides, LPC uses additional lines that allow the slave to inform the master about the desire to send the packet to it. In the Tlink, this information is not required: the master regularly transfers control over the interface to the slave, allowing it to transmit data. Like other interfaces, LPC does not have a credit mechanism.

Thus, the Tlink interface best meets such interface requirements as a minimum of lines, the use of packet transmission and credit mechanism, and symmetry.

Comparing the Tlink interface with other half-duplex protocols, note that packets in the half-duplex Tlink are transmitted in turn: first, exactly one packet is transmitted in one direction, then precisely one packet is transmitted in the opposite direction, and so on. As a result, at full load on the Tlink, the packet transmission rates in each direction, measured in packets per second, will be the same. However, the overall bandwidth of the Tlink interface, measured in bytes per second, will be divided between the directions not equally, but in proportion to the size of the packets.

In the system developed by the authors, this feature is not a disadvantage: on the one hand side, the Tlink will not be fully loaded, and on the other hand side, it is expected that the number of packets transmitted in each direction (per some time) will be approximately the same.

3.7. Measurement

When measuring bandwidth, we will use a frequency of 20 MHz. The maximum theoretical bandwidth of the four communication lines is 9.54 MiB/s.¹⁵

$$\text{Freq} \cdot \text{Width} = 20 \text{ MHz} \cdot 4 \text{ bits} = 80000000 \text{ bits/s} = 9.54 \text{ MiB/s}.$$

We will calculate the bandwidth of the developed Tlink interface under the following conditions: packets containing N bytes (excluding the header) are transmitted only in one direction, packets with data are not transmitted in the opposite direction, only service packets are transmitted.

Then the number of clock cycles required to send the data packet and receive the response will be as follows:

$$(2 \cdot (4 + N)) + 2 + 8 + 2 + 5 = 2 \cdot N + 25,$$

where:

- $(2 \cdot (4 + N))$ — clock cycles required to transmit 4 bytes of the header and N bytes of the packet body over four communication lines,
- 2 — clock cycles required on TAR1 and TAR2,
- 8 — clock cycles required to send the service packet back,
- 2 — clock cycles required on TAR3 and TAR4,

¹⁵MiB — we use binary prefixes to specify the amount of information: 1 MiB = 1048576 bytes.

TABLE 1. Bandwidth

Packet size (bytes)	4	8	16	32	64	124
Bandwidth (MiB/s)	2.31	3.72	5.35	6.86	7.98	8.66
Efficiency (%)	24%	39%	56%	72%	84%	91%

5 — clock cycles required for the signal to pass through the receiving and transmitting registers in the FPGA (4 clock cycles), an additional clock cycle occurs due to the difference in the clock phases of the master and the slave devices.

As a result, the bandwidth will be calculated using the formula:

$$(1) \quad N \cdot \frac{200000000}{2 \cdot N + 25} \text{ B/s} = N \cdot \frac{19.07}{2 \cdot N + 25} \text{ MiB/s},$$

where the first multiplier is the size of one packet (N bytes), the second multiplier is the number of packets sent per second.

For bandwidth measurement, we will use the “observer” block of the Tlink interface, as well as an additional block in the FPGA for generation and receiving of packets of the specified size. When receiving packets, it also checks them for transmission errors.

Bandwidth measurements are presented in the table 1. The first line shows the useful packet size (excluding the packet header).

The second line shows the measured bandwidth. Note that the packet generation and measurements took place directly in the FPGA, without any overhead. Therefore, the bandwidth calculated using the formula 1 is the same as the measured bandwidth.

The third line shows the efficiency of data transmission concerning theoretical bandwidth. For packets with a maximum size of 124 bytes, the efficiency is about 91%, that proofs the efficiency of the implemented approach.

We tested the reliability of the developed Tlink interface at higher frequencies of 66 and 100 MHz. It turned out that to ensure error-free data transmission, it is necessary to use clock signals of the same frequency but with different phases on the master and the slave devices. Therefore, we have added a frequency-locked loop (PLL) block to the master, which changed the phase of the clock signal transmitted to the slave. The specific phase difference depends on the frequency and length of the communication line between the devices. We plan to continue research on the reliable operation of the Tlink at high frequencies.

Conclusion

The development of a computer system based on FPGA set the authors the task of developing a protocol for the interaction of chips with each other with the following requirements:

- support symmetric non-blocking packet data transmission,
- use up to 8 general-purpose input/output lines,
- provide transmission rate at 10 MB/s.

As a result, the authors developed and implemented the Tlink packet protocol, which has the following properties:

- The protocol provides bidirectional packet transmission.
- It uses 6 general-purpose input/output lines (one line intended to transmit the reference frequency, one line to transmit the reset signal, and four lines to transmit data in half-duplex mode). It is possible to use a different number of lines for data transmission by changing parameters of the implementation.
- Packets up to 128 bytes long are supported. It is possible to support packages of other lengths by changing the parameters of the implementation.
- The protocol “knows” about 6 bits in a packet: 5 bits keep the size of the packet, and 1 bit keeps credit information. It is possible to support packages of other lengths by changing the parameters of the implementation.
- The primary transmission frequency is 20 MHz, but frequencies up to 100 MHz are supported (operation at high frequencies is restricted primarily by the quality of the board routing and the performance of FPGA GPIO pins).
- The real transmission rate is up to 8.66 MB/s (transmission efficiency up to 91%). When using a higher transmission frequency, the transmission rate is proportionally higher. When using a protocol implementation with longer packets, the transmission efficiency is higher.

The implemented protocol meets all the requirements of the authors’ FPGA-based computing system. It differs from the considered widely used solutions by the support of packet data transmission.

Authors implemented the protocol in VHDL language and used it in working computing system. You can apply developed protocol to other systems based on FPGA or to specialized chips.

References

- [1] R. Tsugami, Y. Shibata. “FPGA implementation of lightweight communication protocol processing for IoT”, *Proceedings of the 12th International Conference on Complex, Intelligent, and Software Intensive Systems, CISIS-2018, Advances in Intelligent Systems and Computing*, vol. **772**, eds. L. Barolli, N. Javaid, M. Ikeda, M. Takizawa, Springer, Cham, 2019, pp. 506–517. [↑]₅₈
- [2] R. S. Correa, J. P. David. “Ultra-low latency communication channels for FPGA-based HPC cluster”, *Integration*, **63** (2018), pp. 41–55. [↑]₅₈
- [3] R. S. S. Kumari, C. Gayathri. “Interfacing of MEMS motion sensor with FPGA using I2C protocol”, 2017 International Conference on Innovations in Information, Embedded and Communication Systems (ICIIECS) (17–18 March 2017, Coimbatore, India), pp. 1–5. [↑]₆₁
- [4] G. Yoon, J. Kim, G.-Y. Kim, B. Son, H. Yoo. “Multiple RS-485 interface management FPGA design for Power micro-metering”, 2019 10th International Conference on Power Electronics and ECCE Asia (ICPE 2019 — ECCE Asia) (27–30 May 2019, Busan, Korea (South)), pp. 2635–2640. [↑]₆₂
- [5] N. N. Bespalov, Y. V. Goryachkin, D. V. Tundukov. “Software implementation control devices on FPGA for data interchange on SPI interface”, *Scientific and Technical Volga region Bulletin*, 2016, pp. 75–78 (in Russian). [↑]₆₃
- [6] V. S. Starshinov, S. A. Tkachev. “Development of IP-Core for connecting AXI and SPI interfaces using microprocessor systems in conjunction with FPGAs”, *Science. Technology. Innovation*, Proceedings of research papers in 10 parts. V. 1, ed. D. N. Dostovalov, Novosibirsk State Technical University, Novosibirsk, 2018, pp. 110–117 (in Russian). [↑]₆₃
- [7] R. Nawrath, R. Czerwinski. “FPGA-based implementation of APB/SPI bridge”, International Conference of Computational Methods in Sciences and Engineering 2018 (ICCMSE 2018), AIP Conference Proceedings, vol. **2040**, AIP Publishing, 2018, 4 pp. [↑]₆₄
- [8] A. V. Rutkevich, A. A. Veykov, I. Y. Sysoev. “IP-Core SD device development”, *Problems of Advanced Micro- and Nanoelectronic Systems Development (MES)*, 2016, no. 3, pp. 186–190 (in Russian). [↑]₆₄
- [9] M. I. Chemodanov. “Interface organization for realization AXI-Lite protocol over Xilinx FPGA”, *Information technologies in modeling and management: approaches, methods, solutions*, Proceedings of the 1st Russian National Scientific Conference. V. 2, Kachalin Alexander Vasilievich, 2017, pp. 261–268 (in Russian). [↑]₆₆

Received	25.11.2019
Revised	20.03.2020
Published	25.03.2020

Recommended by

dr. Sergey A. Romanenko

Sample citation of this publication:

Igor A. Adamovich, Yuri A. Klimov. “An FPGA packet communication protocol”. *Program Systems: Theory and Applications*, 2020, **11**:1(44), pp. 57–78.

10.25209/2079-3316-2020-11-1-57-78

http://psta.psir.ru/read/psta2020_1_57-78.pdf

The same article in Russian: 10.25209/2079-3316-2020-11-1-31-55

About the authors:

Igor Alekseevich Adamovich

Junior researcher at Aylamazyan Program Systems Institute of RAS. One of the developers of the communication network “Pautina” and the “3D-Torus” communication network of the “SKIF-Aurora” supercomputer.

0000-0001-9728-3024

e-mail: i.a.adamovich@gmail.com

Yuri Andreevich Klimov

Senior researcher at Keldysh Institute of Applied Mathematics of RAS, Ph.D. The developer of the method of program specialization for object-oriented languages by partial evaluation. One of the developers of the communication network “Pautina”, the “3D-Torus” communication network of the “SKIF-Aurora” supercomputer, and the communication software for the SCI network and the “MVS-Express” network of the K-100 supercomputer.

0000-0001-5081-1547

e-mail: yuklimov@keldysh.ru

Эта же статья по-русски:

10.25209/2079-3316-2020-11-1-31-55