# VerCoLib: Fast and Versatile Communication for FPGAs via PCI Express

Oğuzhan Sezenlik*, Sebastian Schüller*, and Joachim K. Anlauf
*These authors contributed equally to this work*

University of Bonn,
Institute of Computer Science VI, Technical Computer Science,
Endenicher Allee 19 A, 53115 Bonn
`sezenlik@cs.uni-bonn.de`
`schueller@ti.uni-bonn.de`
`anlauf@cs.uni-bonn.de`

**Abstract.** PCI Express plays a vital role in including FPGA accelerators into high-performance computing systems. This also includes direct communication between multiple FPGAs, without any involvement of the main memory of the host. We present a highly configurable hardware interface that supports DMA-based connections to a host system as well as direct communication between multiple FPGAs. Our implementation offers unidirectional channels to connect FPGAs, allowing for precise adaptation to all kinds of use cases. Multiple channels to the same endpoint can be used to realise independent data transmissions. While the main focus of this work is flexibility, We are able to show maximum throughput for connections between two FPGAs and up to 88% saturation of the available bandwidth for connections between the FPGA and the host system.

**Keywords:** VerCoLib, PCI Express, FPGA, Communication Library, Transceiver

## 1 Introduction

FPGAs are widely used to accelerate state-of-the-art algorithms or as co-processors in heterogeneous high performance computer systems. FPGA vendors offer affordable evaluation boards with high-end FPGAs, especially popular in academic research. Through the development of high level synthesis tools like Xilinx Vivado HLS or intelFPGA OpenCL, FPGAs became a more accessible and viable platform. While writing code for the FPGA accelerator itself is one part of a design another important factor is to utilise its full performance by transferring data reliably and with sufficient throughput. Here a common high-bandwidth interface like PCI Express (PCIe in the following) is essential, the use of which requires fundamental knowledge about its protocol, underlying computer hardware as well as kernel driver programming. Therefore developers often face the problem to implement the required complex logic to interface the PCIe IP core provided by the vendor and integrate access to the accelerator into their software. Furthermore modern mainboards allow devices to communicate directly via PCIe, completely bypassing the main memory, a feature heavily used to connect multiple GPUs and build low-cost supercomputers for various scientific applications. The same idea also applies to FPGAs: one could simply plug several off-the-shelf FPGA boards into one standard desktop computer to improve the computational power. Such a feature is especially useful, since combining several smaller FPGAs is more cost efficient than using high-end variants.

Our goal is to provide a highly configurable and easy to use open-source communication library (VERsatile COmmunication LIBrary) that allows FPGA-programmers to focus on their primary objective, namely implementing their algorithms. With our interface it is very easy to build affordable multi-FPGA computers, where communication is performed between Host and FPGA as well as directly between FPGAs via PCIe. This includes configurable and generic modules used on the FPGA as well as a Linux kernel driver to set up communication channels and performing the host part of the data transmission. After setting up the channels between FPGAs, the host system is not involved into FPGA-FPGA communication at all.

This paper is structured as follows: In section 2 we give an overview about existing commercial and open-source PCIe solutions and our motivation for the development of VerCoLib. Then the hardware architecture and features of our transceiver are described in section 3, followed by the software interface and driver in section 4. Finally the resource consumption and performance are evaluated in section 5.

## 2 Related Work

There are already several other systems that provide a PCIe interface for FPGA accelerators. In general the different solutions can be categorised based on the PCIe configurations they support and the resulting theoretical maximum bandwidth. This bandwidth depends on the generation of the PCIe standard as well as the number of PCIe lanes a device is connected to. For reference, the maximum bandwidth for a Gen2 device with 8 lanes is 4 GB/s. This also applies to Gen3 devices with 4 lanes. Gen3 devices with 8 lanes theoretically reach a bandwidth of 8 GB/s.

Out of the available commercial solutions, the interfaces provided by Xillybus [1] and Northwest Logic [2] are the most notable ones. The designs from Northwest are limited to Xilinx devices and are used in the reference design Xilinx offers, supporting PCIe Gen3 with 8 lanes. Xillybus is available for both Xilinx and intelFPGA FPGAs and provides host software for Linux and Windows operating systems.

Academic solutions include RIFFA 2.2 [3], JetStream [4], ffLink [5], EPEE [6] and DyRact [7]. Out of these solutions, PCIe Gen2 is supported by EPEE (8 lanes), DyRact (4 lanes) and RIFFA (8 lanes), Gen3 is supported by RIFFA (4 lanes), JetStream and ffLink (8 lanes).

RIFFA offers a host-FPGA interface for a wide range of devices and PCIe configurations with drivers for both Linux and Windows as well as APIs for a variety of programming languages. Their transceiver reaches a throughput of up to 3.64 GB/s (upstream) and 3.43 GB/s (downstream).

EPEE is designed around the concept of a general purpose PCIe interface including DMA communications with up to 3.28 GB/s, a set of IO registers reachable from both hardware and software as well as user defined interrupts. They support Xilinx Virtex-5,6 and 7 series FPGAs on Linux systems. While multiple independent DMA channels are supported as a plugin, there are no measurements of the performance impact in terms of resource usage or throughput.

DyRact implements an interface for dynamic partial reconfiguration within its PCIe solution, allowing for convenient and efficient reconfiguration of user designs with PCIe connections. Since they only support Gen2 devices with up to 4 lanes, their bandwidth peaks at 1.54 GB/s.

The ffLink interface is created mainly out of IP-Cores supplied by Xilinx and relies heavily on the AXI-4 [8] infrastructure. Strongly relying on IP-Cores has the advantage of low development times and bug fixes from the IP-Core developer. On the flip side, this also means a comparatively high resource usage and makes it impossible to adapt the design to other vendors. The ffLink system achieves a maximum throughput of 7.04 GB/s.

JetStream is the only other solution we have found that supports direct PCIe FPGA-FPGA communication. They demonstrate the effectiveness of direct FPGA-FPGA communication with a large FIR filter that spans multiple FPGAs. They were able to show that distributing the data directly between FPGAs can result in a reduction of memory bandwidth by up to 75%. Their host-FPGA solution supports only Gen3 devices with 8 lanes with a maximum throughput of 7.09 GB/s. However, the missing support for Gen2 devices effectively limits the use of JetStream to high-end devices.

## 3 FPGA Transceiver Design

The central concept of VerCoLib deals with unidirectional, independent channels. A channel is the user interface to send or receive data, translating between raw data and PCIe packets.

Every configuration of the transceiver has the same structure, consisting of one endpoint module, an arbiter and an arbitrary number of channels, all of them using the same handshake interface, equivalent to AXI4-Stream [8]. An example is shown in Fig. 1.

The function of the endpoint is to handle global resources which need to be shared among the channels. This includes interfacing the Xilinx specific hard IP-Core [9] and handling internal communication with the software driver as well as managing the interrupts from all channel modules and providing dynamic tag mapping for downstream DMA transfers.

We are using MSI-X interrupts that allow devices to allocate up to 2048 interrupt vectors instead of only 32 vectors allowed by MSI. This makes it possible

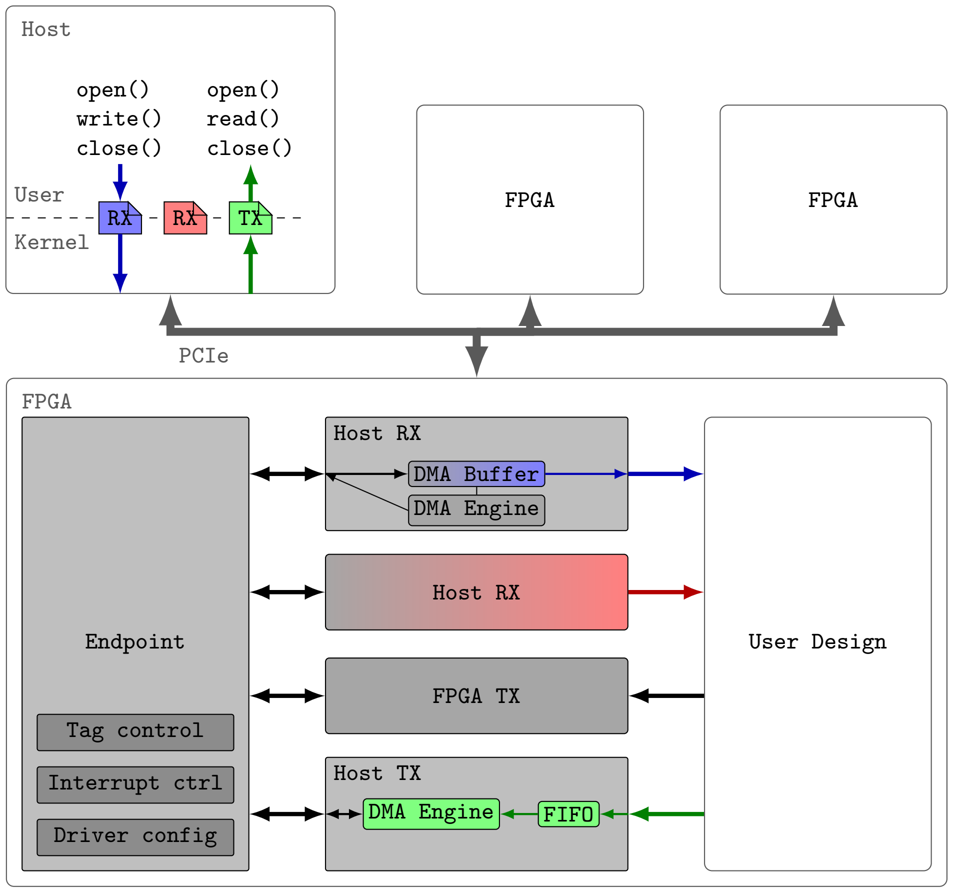

**Fig. 1.** System overview of example configuration. Best viewed in color. The colors indicate independent data streams. Black and gray connections may contain data from all streams and fading colors displays the data being filtered. Note that a user can instantiate channel modules as necessary.

to map every channel uniquely to a MSI-X vector. Thereby the driver is able to immediately identify the channel that issued an interrupt without further communication, which in turn reduces communication overhead and latency.

In a multi-channel PCIe transceiver design special consideration is required when handling DMA transfers from host to FPGA. According to the PCIe specification [10], the receiver has to request data from the main memory of the host system with memory read request packets. These are answered by the host via memory completion packets with the requested data as payload. PCIe provides a tag field with at most 256 different values in the packet header which is used to determine the affiliation of each completion with a request. To reach optimal utilization of the PCIe-bandwidth the transceiver has to keep several requests in flight. The easiest solution to identify the channel a completion belongs to would be to use a fixed range of tag values per channel, but in this case the number of available tag values per channel depends on the total number on instantiated channels. This could reduce the bandwidth a single channel can reach, which contradicts our goal to provide a flexible and high-performance interface, without making assumptions about the detailed requirements of the attached user design. In our solution the endpoint assigns unused tags dynamically to requests. Note that no demultiplexing is done anywhere in the design. Instead all channels receive the same data, filter out all packets not belonging to them and ignore the rest. As a result the design allows for multiple independent data streams by simply attaching more channel modules to the same endpoint with a simple arbiter module. Channels may be arbitrarily mixed to optimally fit the requirements of the application, while using as little as possible of the resources of the FPGA.

Everything else is handled by the channel modules. They are designed individually for specialised tasks, translating between raw data and PCIe packets, as shown in Fig. 1. At this point we have implemented four different channel types (host-rx/tx, FPGA-rx/tx). Each channel takes complete care of a unidirectional transfer to or from the host or the FPGA.

## 3.1 Host communication channel

All data transfers between host and FPGA are initiated by the software driver. The host-rx channel implements the downstream direction of the host communication, i.e. the transmission of data from host to FPGA. First the driver sends the address and size of the data buffer to be transmitted to the host-rx channel with the desired id. Then the DMA-Engine will check the available free memory in the DMA-Buffer and starts requesting data from the main memory of the host. Note that PCIe permits the host to answer requests in a different order than originally issued. The DMA-Buffer is capable of reordering incoming completion packets in order to extract and forward the raw data to the user design attached to it. After the transfer is finished the DMA-Engine issues an interrupt, which is finally handled by the MSI-X-interrupt handler in the endpoint and waits for the next buffer information.

The host-rx channel is capable of processing incoming packets every clock cycle, therefore it never stops the endpoint from sending data. Since it requests only as much data as it can store in its memory, it will never create back-pressure to the endpoint. This guarantees that deadlocks from data transmissions are avoided in all cases. The throughput depends on the rate the user design pulls data from the channel and the available bandwidth provided by the PCIe bus.

The host-tx channel works similar in the upstream direction. As before the DMA-Engine waits for the buffer information from the driver and checks if the FIFO has gathered enough data from the user design to generate at least one memory write request packet. While the DMA-Engine always tries to maximise the size of a packet to ensure high transfer rates, the user design has the option to force a transmission prematurely by tagging the last valid data word it presents to the host-tx channel. This is necessary in cases where the user design does not produce enough data to fill a PCIe packet completely. Furthermore the channel is able to generate and transmit PCIe packets at every clock cycle, as long as the vendor-ip-endpoint is capable of accepting data.

### 3.2 Direct FPGA-FPGA Communication

To realise direct FPGA to FPGA communication it was sufficient for us to develop a variation of the host communication channels. In order to send data unidirectionally from one device to the other, a pair of FPGA-FPGA channel modules is necessary: one receiver channel on the receiving FPGA to request data and buffer incoming packets and one sender channel on the sending FPGA to pack the PCIe packets and send them off.

These pairs are connected by the software on the host system which sends the necessary address information to each channel during initialisation of the system. The addresses are global on the complete system and consist of the FPGA PCIe bus- and device number as well as a local channel number which needs to be unique on its FPGA design. Pairs can be re-assigned by the software at any point in time. For the sake of simplicity, we have not implemented a mechanism that blocks re-assignment during a transfer, meaning that the user should check that all communication ceased before changing the pair configurations.

Since all channel modules are designed never to create back-pressure to the endpoint, the receiving channels contain a FIFO storing incoming packets. To ensure that this FIFO never overflows, the receiver actively requests an amount of data from the sender that is guaranteed to be free in the FIFO. If the user design consumes enough data from the FIFO, the receiver can safely post a new request even if not all data from the last request arrived yet. The sender combines all requests it receives into a single one.

To minimise the overhead produced by PCIe packet headers, the sender channel tries to create PCIe packets of maximum size. Smaller packets will only be created if the user design indicates the end of the data stream or the amount of requested data is too small.

In contrast to the JetStream design in [4] which employs memory read request and completions to realise direct FPGA to FPGA communication, we chose to use a protocol solely based on memory write requests. Vesper et al. advocate their decision with the ability to reuse the already existing host communication for direct FPGA-FPGA transmissions. On the other hand, creating special modules allows to save a lot of FPGA resources since there are fewer cases to take care of when transferring data between FPGAs. The issue of memory reordering is one example of a fairly complex and resource intensive operation that can be ignored, since we are in control of both endpoints.

Although the presented channels are sufficient for most FPGA-accelerators, some designs might require special functionality and therefore modifications to the transceiver. In order to add features, understanding existing open-source transceivers is often very difficult, hence modifying them is an error prone task. The modularity of our transceiver allows the user to develop and integrate their own channels without the necessity to alter the already existing infrastructure. The developer only has to make sure, that the new channel implements the handshake-protocol to the endpoint and arbiters correctly.

In fact, the first version of the transceiver only allowed host to FPGA communication, channels, enabling direct FPGA to FPGA connections were added later, without changing the overall architecture.

Another benefit of our design can be shown with the following use case. Imagine a developer wants to prioritise transfers higher for one particular core he has implemented. This is simply possible by replacing the arbiter with a weighted version. The arbiter itself is a very simple module, agnostic to the PCIe interface and therefore it contains no special logic to process PCIe packets.

## 4 Software Interface

The Linux kernel driver for VerCoLib focuses on simplicity and tries to adhere to standard Unix syscalls.

The driver presents each hardware channel as its own device which can be interacted with using standard calls like open, close, read, and write. On initialization the driver gets the number and directions of instantiated buffers automatically from the FPGA transceiver. Note that hardware receiver channels only support the 'write' call and analogue to this hardware sending channels only support the 'read' call.

Aside from that the behaviour of the channel interfaces is comparable to communication through a POSIX network socket. After opening a channel device the user only needs to supply a pointer to the buffer that should be read from/written to, as well as the number of bytes that should be transferred.

Internally the driver manages up to eight buffers per channel to realise multi buffering for enhanced performance. If the user issues a transfer consisting of more bytes than the buffers can hold, the driver will copy as many data as possible into the buffers and informs the user of the amount of copied data. The write/read calls return immediately after copying and remember used buffers internally. If the user wants to transmit data on a channel that has no free buffer available the call will block until the current transmission is finished. The driver supports the poll/select calls on channel devices to allow the user to find a channel ready for transmissions. For transmissions from the FPGA to the host, the driver continuously requests data from the host-sender channel as soon as the channel device is opened by the user. This allows for data transmissions before the user reads from the channel device and thus reducing the latency of this operation.

Since each channel has an individual set of buffers, the channel devices are inherently thread-safe. Thus the user is allowed to handle different channel devices in individual threads/processes.

Fig. 2 shows a simple but representative use of the software API. Since the API follows the POSIX standard it automatically supports all languages (additional to C/C++) that allow for direct write/read calls, e.g. python or rust.

The driver creates one additional device per FPGA in order to allow the user to communicate directly with the hardware PCIe transceiver. This interface offers the user the ability to set up and control direct FPGA-FPGA communication between channels on different FPGAs. This is also very useful during development to debug the state of the hardware.

```
unsigned long size = data_transfer_size();
void* buffer = create_data_to_transfer();

int fd = open("/dev/fpga_0_downstream_0", O_WRONLY);

while(size) {
    unsigned long written = write(fd, buffer, size);
    if(written < 0) handle_error();
    buffer += written;
    size   -= written;
}

close(fd);
```

**Fig. 2.** API of the software interface

## 5 Evaluation

For all experiments in this section, we used two Xilinx VC707 development boards (containing the Virtex-7 XC7VX485T) which are communicating with 8 PCIe Gen2 lanes. The host system features a Intel Core i5-3300 CPU and 16 GB of RAM. DyRact and RIFFA also took their measurements on the XC7VX485T board while ffLink and JetStream used the Xilinx VC709 development board with the Virtex-7 XC7VX690T chip. It should be noted that the VC709 supports the PCIe Gen3 standard, leading to the expectation of higher throughput and a higher absolute resource usage.

In a typical scenario with one data connection from the host to the FPGA and one connection in the opposite direction, the VerCoLib PCIe transceiver uses very few FPGA resources as shown in Table 1. Only DyRact used fewer resources than VerCoLib and they only implement a Gen2 solution with up to 4 lanes which is implemented internally with a 64 b interface in contrast to the 128 b interface used by Gen2 solutions with 8 lanes. If we additionally consider having direct FPGA to FPGA communication only, our solution uses the fewest resources overall.

In Table 2 we compare configurations with multiple independent data channels. The notation VerCoLib ($x$,$y$) should be read as a PCIe configuration with $x$ host downstream and $y$ host upstream channels. The resource consumption of the RIFFA configurations were calculated from the costs of adding a single channel to the configuration, reported in [3]. It can be seen that our solution uses significantly fewer resources than a comparable configuration of RIFFA. Especially configurations which use an asymmetrical amount of channels such as VerCoLib (6,1) show the benefit of uni-directional channel designs. A RIFFA configuration that allows for six independent connections uses significantly more resources since it only allows for bi-directional connections.

**Table 1.** Comparison of resources of different PCIe solutions. [*] ffLink provided measurements given in percent only. The absolute numbers were calculated based on the data-sheet of the VC7VX690T.

| | LUT | FF | BRAM |
|---|---|---|---|
| DyRact | 5181 | 6971 | 26 |
| RIFFA 2.1 | 7396 | 7489 | 16 |
| VerCoLib Host | 6410 | 7817 | 11 |
| VerCoLib FPGA | 5086 | 6559 | 8 |
| ffLink* | 12996 | 43320 | 132 |
| JetStream | 8571 | 6955 | 17 |

**Table 2.** Resource comparison of configurations with multiple channels.

| | LUT | FF | BRAM |
|---|---|---|---|
| RIFFA 2 channel | 11767 | 12710 | 28 |
| RIFFA 6 channel | 29251 | 33594 | 76 |
| VerCoLib (2,2) | 8524 | 10007 | 17 |
| VerCoLib (6,6) | 18276 | 21021 | 41 |
| VerCoLib (6,1) | 12778 | 15602 | 31 |

For all tests the maximum payload size was set to 128 bytes. A larger maximum payload size would have been preferable, however the available host system did not allow for 256 or more bytes.

With a maximum of 128 B payload and 16 B header we can reach up to $128/(128+14) \approx 0.888$ times of the bandwidth the wire can handle. Since PCIe Gen2 with 8 lanes can reach a throughput of 4 GB/s, the maximum throughput we can achieve in practice is $\approx 3.555$ GB/s.

**Table 3.** Maximum transfer rates in GB/s for host - FPGA communication. Omitted half duplex measurements in asymmetric configurations are equal to the respective symmetric ones. ([*] downstream/upstream)

| Configuration | half duplex downstream | half duplex upstream | full duplex |
|---|---|---|---|
| DyRact | 1.54 | 1.51 | N/A |
| VerCoLib (1/1) | 3.13 | 3.11 | 2.78 |
| VerCoLib (4/4) | 3.13 | 3.18 | 2.79 |
| VerCoLib (4/1) | | | 2.8 |
| VerCoLib (1/4) | | | 2.79 |
| RIFFA 2.1 | 3.32 | 3.56 | N/A |
| EPEE | 3.2 | 3.28 | 2.76/2.62[*] |
| ffLink | 7.1 | 6.3 | N/A |
| JetStream | 6.4 | 6.4 | N/A |

Bandwidths for host to FPGA data transmission with different channel configurations are shown in Table 3. We measured the time a test software takes to stream 1 GB of data in each direction. In half-duplex mode our transceiver achieves transfer rates of about 3.13 GB/s for downstream (host to FPGA) and 3.11 GB/s for upstream transfers, thus utilizing about 88% of the theoretical maximum in both directions. In full duplex mode VerCoLib achieves a throughput of 2.79 GB/s, or 79% of the theoretical maximum in both directions. Different channel configurations, including asymmetric designs have no noteworthy impact to the performance.

While RIFFAs implementation achieves about up to 11.4% higher throughput than VerCoLib in half duplex mode - at the cost of considerable more resource consumption as shown before - they did not publish comparable values for full duplex transfers. EPEE shows very similar results in terms of throughput, whereas the former is slightly faster in half-duplex mode, VerCoLib has an advantage in full duplex transfers. Although the performance of our design is not directly comparable to DyRact, ffLink and Jetstream, all show similar transfer rates with respect to the theoretical maximum given by the different hardware they are using.

For direct communication between FPGAs we performed measurements of the time it takes to transfer 16 GB data sequences. The measurements were taken by counting the cycles between the first and last data word on the target FPGA. This kind of measurement implies that the initial latency between sending the first request for data and the arrival of the data is not taken into account. For a sufficiently large amount of transferred data, the influence of this latency is not significant. The measurements were performed over different amounts of transferred data, ranging from 4096 kbyte to 16 GB. The measured mean transfer rate was 3351.45 MB/s, which is roughly 204.1 MB/s less than the estimated theoretical throughput.

Additional measurements were obtained on a transceiver-configuration using two and four channel pairs. The setup for these were taken analogue to the first measurement, each channel transferred a 16 GB long data sequence to the target FPGA. This resulted in a mean transfer rate of $\approx$ 1675 MB/s per channel for the configuration with two channel pairs and $\approx$ 837 MB/s per channel for the configuration with four pairs. The total transfer rate per configuration adds up to $\approx$ 3351 MB/s in both cases, implying that using multiple channel pairs has no negative effect on the bandwidth. Since all channel pairs show similar results in every configuration, it can be further argued that the channels indeed work independently from each other.

A separate experiment was conducted to measure the round trip time of a single packet from one FPGA to another and back again. For this a special module was written which sends a packet of length one to the other FPGA and counts the number of cycles it takes until the packet returns. These counts are summed up over a series of 2000 sent packets, resulting in a mean round trip time of $\approx$ 176.5 cycles or $\approx$ 705 ns with a standard deviation of less then 0.15 cycles or 0.6 ns.

## 6 Conclusion

VerCoLib is a versatile, fast, and resource saving framework for communication from FPGA to FPGA, from host to FPGA, and FPGA to host via PCIe. The flexibility mainly originates from the fact that the logic for generating PCIe packets and reacting on responses from the PCIe bus is handled individually by the channel modules. Nevertheless the resource consumption in typical situations is not higher than in other solutions since the configuration can be adapted to the requirement of the applications in a very flexible way, using absolutely needed resources only. Standardised interfaces allow for straightforward extensions of functionalities and reduce the effort necessary for maintenance.

Part of the framework is a Linux kernel driver that is responsible for setting up the communication channels and performing the communication between host and the FPGAs, whereas the communication between FPGAs is handled completely by the hardware itself without any interaction with the driver or the application software, except for the initial setup of the channels. VerCoLib was tested on the Xilinx Virtex-7 VC707 development board and the Linux kernel version 4.4.

## 7 Future Work

We plan to release the VerCoLib framework under a permissive open-source license in the near future. Without changing the general philosophy of VerCoLib it will be possible to integrate other communication channels into the same framework, e.g. Ethernet or USB channels. Thanks to the modular design, we are confident that VerCoLib is easy to adapt to other hardware, e.g. an FPGA using the PCIe v3.0 standard. Another topic for future work is extending VerCoLib to support scatter/gather DMA transmissions.

## References

1. Billauer, E.: Xillybus. URL `http://xillybus.com`
2. Northwest Logic: PCI Express Solution (2017). URL `http://nwlogic.com/products/pci-express-solution/`
3. Jacobsen, M., Richmond, D., Hogains, M., Kastner, R.: Riffa 2.1: A reusable integration framework for fpga accelerators. ACM Trans. Reconfigurable Technol. Syst. **8**(4), 22:1–22:23 (2015). DOI 10.1145/2815631. URL `http://doi.acm.org/10.1145/2815631`
4. Vesper, M., Koch, D., Vipin, K., Fahmy, S.A.: Jetstream: An open-source high-performance pci express 3 streaming library for fpga-to-host and fpga-to-fpga communication. In: 2016 26th International Conference on Field Programmable Logic and Applications (FPL), pp. 1–9 (2016). DOI 10.1109/FPL.2016.7577334
5. de la Chevallerie, D., Korinth, J., Koch, A.: fflink: A lightweight high-performance open-source pci express gen3 interface for reconfigurable accelerators. SIGARCH Comput. Archit. News **43**(4), 34–39 (2016). DOI 10.1145/2927964.2927971. URL `http://doi.acm.org/10.1145/2927964.2927971`
6. Gong, J., Wang, T., Chen, J., Wu, H., Ye, F., Lu, S., Cong, J.: An efficient and flexible host-fpga pcie communication library. In: 2014 24th International Conference on Field Programmable Logic and Applications (FPL), pp. 1–6 (2014). DOI 10.1109/FPL.2014.6927459
7. Vipin, K., Fahmy, S.A.: Dyract: A partial reconfiguration enabled accelerator and test platform. In: 2014 24th International Conference on Field Programmable Logic and Applications (FPL), pp. 1–7 (2014). DOI 10.1109/FPL.2014.6927507
8. ARM Ltd.: AMBA AXI4-Stream Protocol Specification v1.0 (2010). URL `http://infocenter.arm.com/help/index.jsp?topic=/com.arm.doc.ihi0051a/index.html`
9. Xilinx Inc.: 7 Series FPGAs Integrated Block for PCI Express v2.2 Product Guide for Vivado Design Suite (2016)
10. PCI-SIG: PCI Express Base Specification Revision 2.1 (2009)