JOINT INSTITUTE FOR NUCLEAR RESEARCH
Meshcheryakov Laboratory of Information Technologies

FINAL REPORT ON THE START PROGRAMME

Development of L2 hub SPD, NICA

Supervisor:
Mr Viacheslav Tereshchenko

Student:
Dmitriy Erofeev, Russia
Tomsk State University

Participation period:
July 01 – August 10,
Summer Session 2024

Dubna, 2024

Contents

1. NICA project and SPD

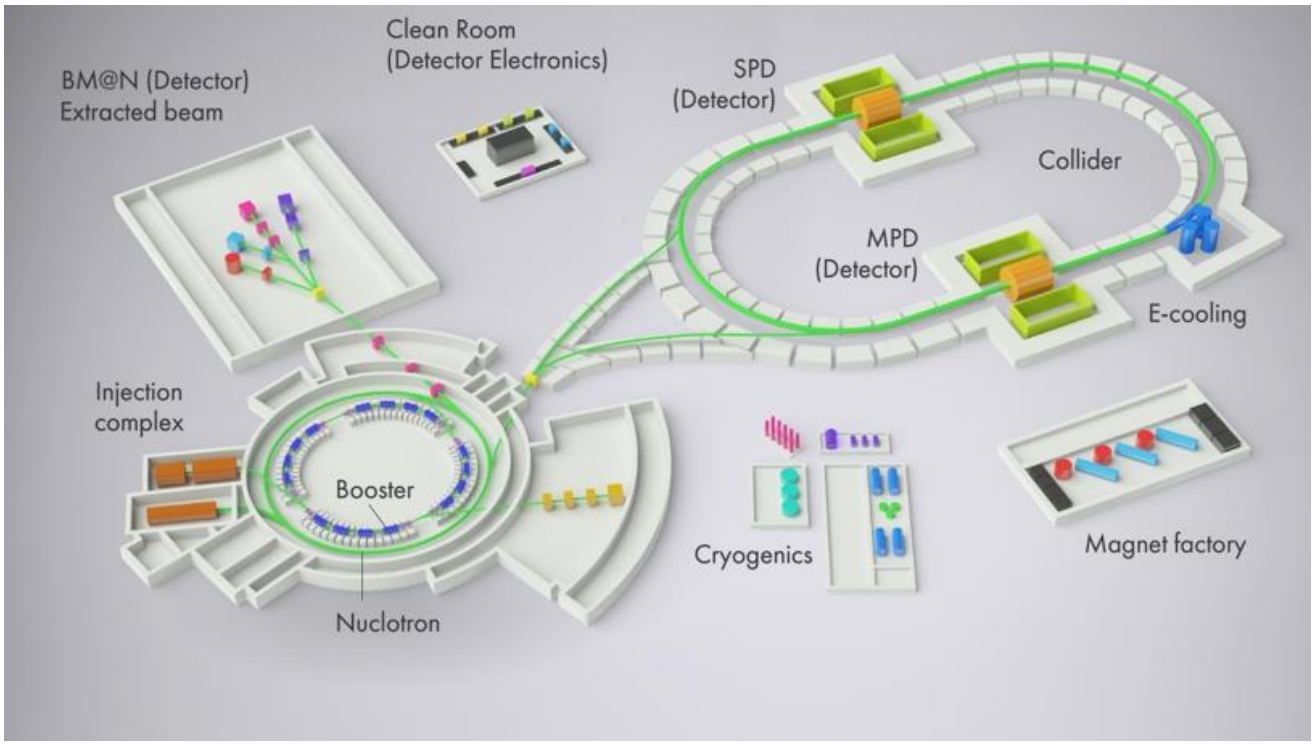

Fig. 1. Schematic view of the NICA-Nuclotron complex and the position of the SPD.

The Nuclotron-based Ion Colider fAcility (NICA) is currently under construction at the Joint Institute for Nuclear Research [1]. The global aim of this project is to explore the phase diagram of strongly interacting matter in the region of highly compressed and hot baryonic matter. The SPD is the experiment of the NICA accelerator complex (Fig.1.) The aim of SPD is the to study the spin structure of the proton and deuteron and the other spin-related phenomena with polarized proton and deuteron beams at a collision energy up to 27 GeV and luminosity up to 1032 cm−2 s−1. The detailed description of the SPD experiment can be found in Ref. [2]

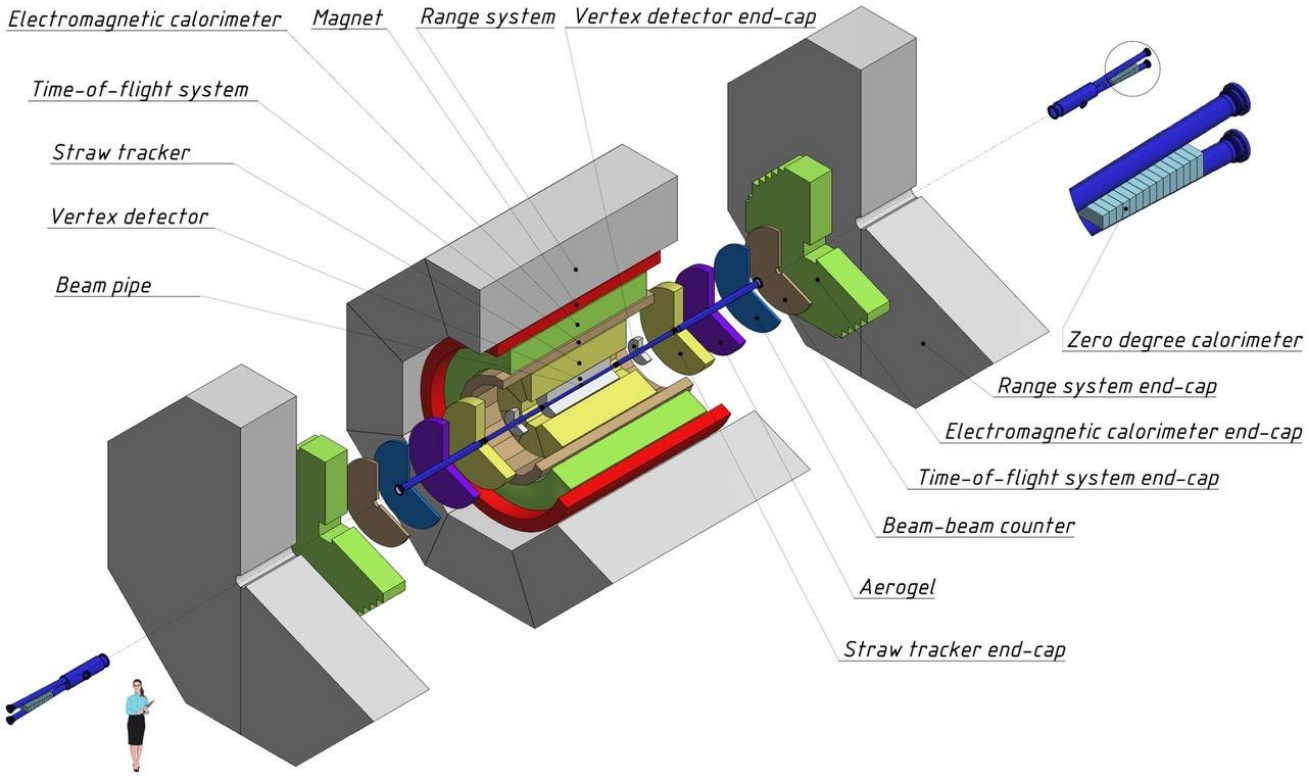

Fig. 2. Scheme of SPD detector.

The general view of the SPD detector is shown in Fig.2. The detector, as one of the main components of the experiment, is designed to process the information generated after particle collisions. This involves signal processing—first analog, then digital. One of the final stages of processing this signal is the L2 hub. This is an FPGA designed to quickly and losslessly transmit important information to the computer.

2. Z19-P

FPGA (Field-Programmable Gate Array) is a programmable logic integrated circuit. It is a chip that differs from a processor in that the logic of such a chip can be configured according to the user's needs. A processor is a fairly universal and powerful chip that performs well in the subsequent stages of SPD (Frontend); however, in specialized tasks such as big data processing or machine learning, the resources of the processor may not be sufficient. In our case, it is also beneficial to use an FPGA—the configuration of this chip allows its design to focus on performing one task: transferring data to the computer intact and unaltered, without delving into the essence of that data.

In our project, we are using the Alinx Z19-P (Fig. 3) board, which is specialized for PCIe and Ethernet. Figure 3 shows its appearance without the SFP cage (Which accepts data). The PCIe is of the fourth generation (Gen 4.0) and has a bus width of x16. In theory, such PCIe can deliver speeds of up to 16 GB/s, which I will need to study in my subproject.

Fig. 3. FPGA Xilinx - Alinx Z19-P.

2.1 L2 hub - General overview of the project

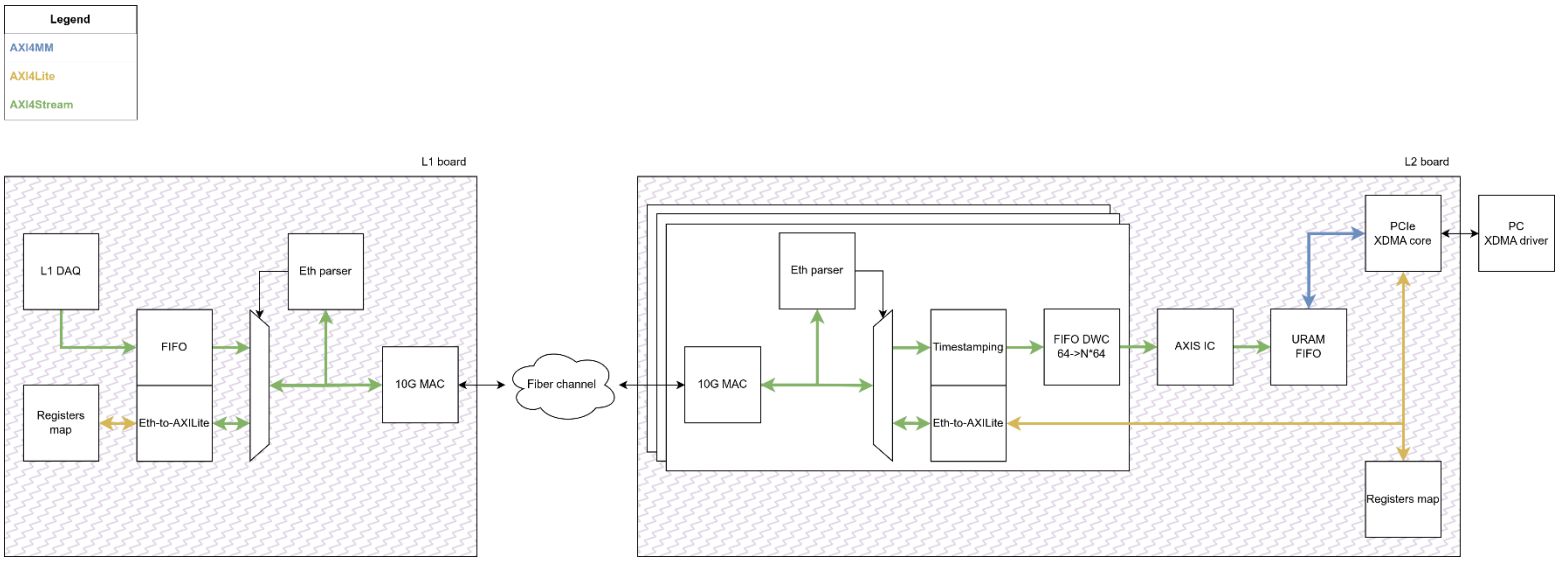

Fig. 4 Functional diagram of L1 and L2 concentrator.

The general principle of operation of the L1-L2 (Fig. 4) concentrator is as follows (from left to right):

Packetized data from the L1 DAQ module is transmitted to a FIFO buffer and then to an arbiter (Multiplexer) – its task is to determine from the headers what type the packet belongs to – command or data.

EthParser is a module that breaks down packets into their components (Header, data, etc.). If it is data, the packet is then forwarded to the 10G MAC module, which, in turn, along with the 10G MAC module on the L2 concentrator, checks the MAC addresses of the packets (there are many L2 boards, and each has its own MAC). The multitude of L2 boards is necessary to accelerate data transmission – one of the main functions of L2 and L1.

If the packet is a command (a command can be stopping or starting the design operation), it will be sent to the register map via the Eth-to-AxiLite protocol converter (since the register maps only operate on AxiLite).

On L2, the situation is exactly mirrored, but before the FIFO stands Timestamping – a module that adds the time of packet processing for monitoring throughput and synchronization.

After that, packets are transmitted to the DWC FIFO – a FIFO buffer in frame mode with adjustable data sizes, which allows for accumulating them and sending large reports directly to the PC through the XDMA IP core.

This project is divided into two parts. The first part is Ethernet, which is handled by my colleague Andrey Berngardt, while I am responsible for the second part—PCIe. In general terms, our

plan for our stay in Dubna is divided into two levels:

1. Minimum plan. This includes: Installing software on the computers, configuring them, and setting up the board. We need to bring up the interfaces since we previously worked on a different board with PCIe Gen1x4 and 1G Ethernet, and now we have PCIe Gen4x16 and 10G Ethernet.

2. Intermediate plan. This includes: Connecting our designs, generating a packet to optical fiber, receiving it from that fiber, and routing it through the entire design, then displaying it on the computer.

3. Maximum plan. Measure the speed of PCIe, connect the L1 board to the L2 board, receive a packet from the L1 board, and see it on the computer.

3 Realization

3.1 Plan Minimum and Intermediate Plan

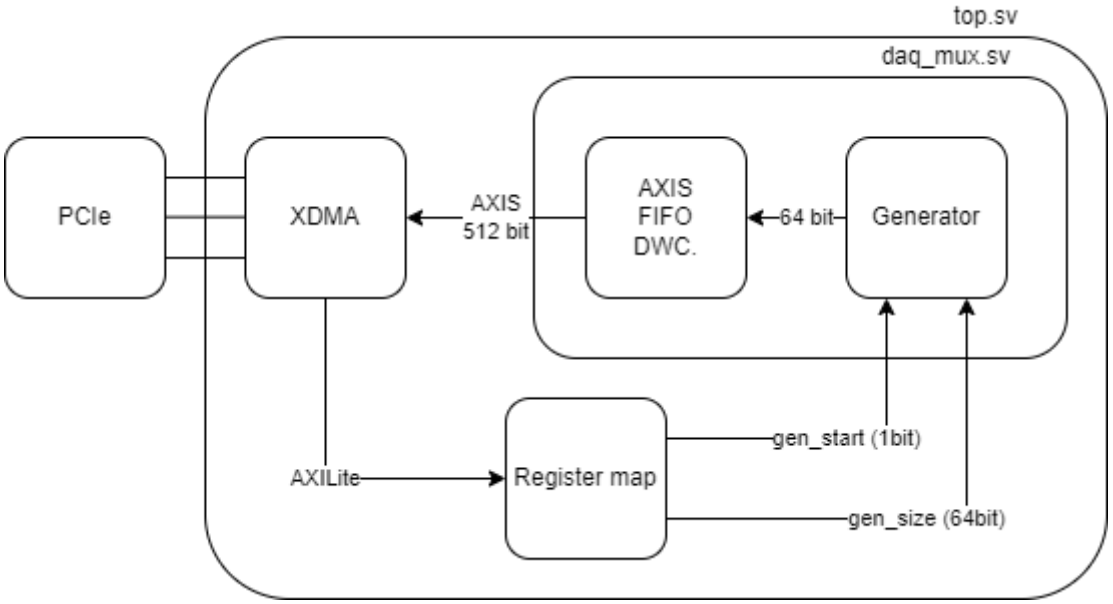

Fig. 5 Functional diagram of L1 and L2 concentrator.

The main task of my part of the project is to prepare the "groundwork" for the overall design to accept packets from the PCI side. The general concept is as follows:

The XDMA module is an IP core for interacting with the design. Using it and the AXI Lite interface, I send a command to the register map to start the packet generator. Then, the packet generator fills the FIFO buffer. After that, I read the data from it using XDMA. This diagram shows the updated version of the design for Gen3x16. With it, I can successfully test XDMA and PCIe, and study the bandwidth with different packet sizes.

After that, we proceeded to combine our designs (Andrei's design was inserted instead of the generator). By slightly modifying the design for proper operation in tandem, we received a packet on the computer, as shown in Figure 6. It contains 512 bytes of data + 1 byte of service information (packet size validity, checksum, and timestamp).

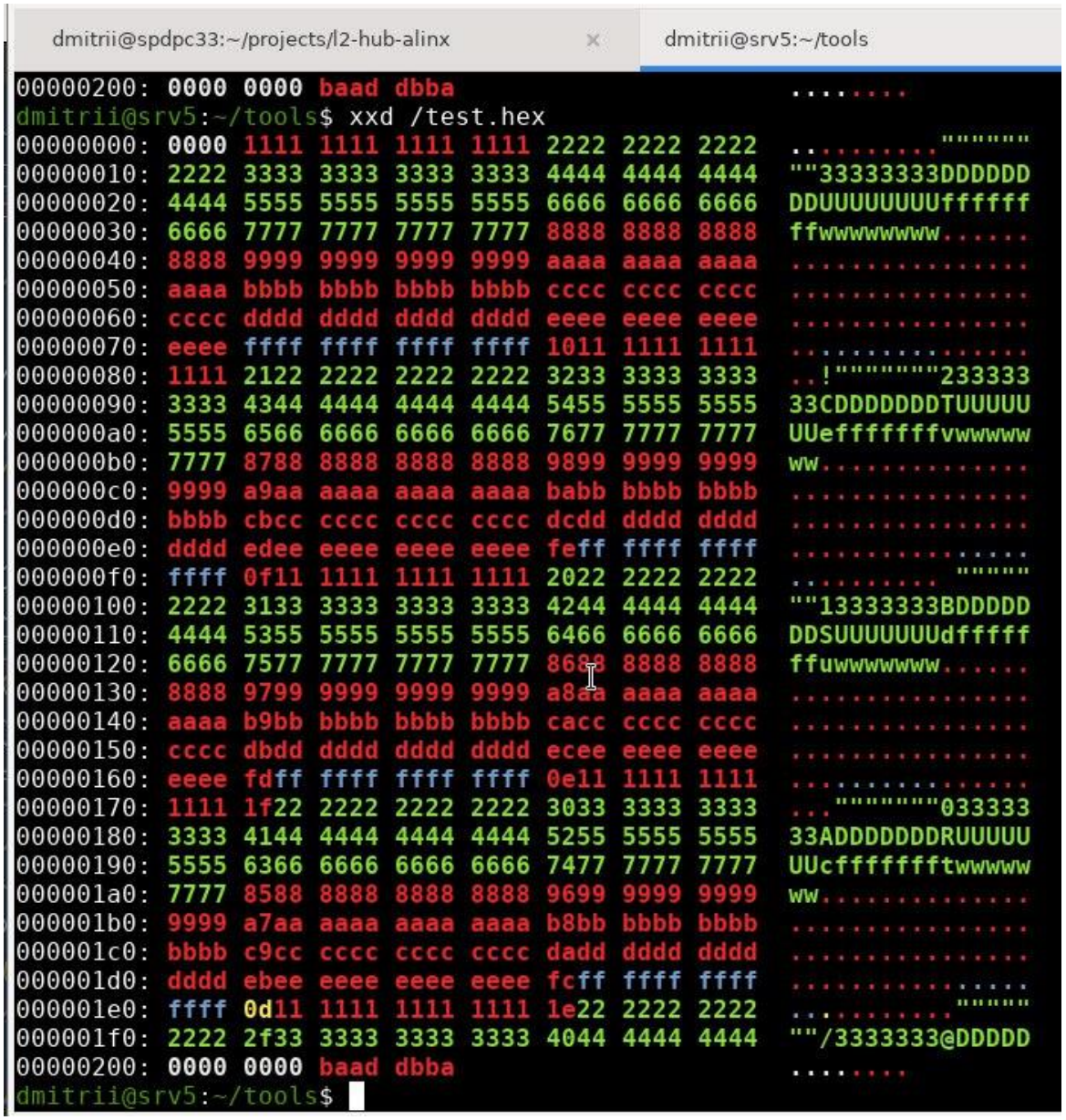

Fig. 6 Packet on the computer

3.2 Plan Maximum

Then I started measuring the PCIe speed using my design. This is necessary so that we know what kind of performance we can expect in the experiment. However, I encountered a problem: the speed was only about 37% of the maximum theoretical speed. Our system is Fedora 39, Linux Kernel 6.5.2. Motherboard is Supermicro H12SSL-NT and CPU is AMD EPYC 7402P 24-core. As a result of extensive work, I have tried absolutely all debugging options available to me, but the speed remained the same. Here are my observations and assumptions:

1. The processor is not fully utilized. The maximum is 60%. I find this strange because, for example, the command dd of=/dev/null if=/dev/zero bs=1MB count=10000 fully loads the processor at 100%, while dd of=/dev/null if=/dev/xdma0_c2h_0 bs=1MB count=10000, which also transfers bytes via XDMA (I checked), also shows 60% and the same bandwidth. This detail is one of my arguments for why the programs I am using (like dma_from_device.c, etc.) are not related to speed limitations; rather, it is the driver that limits it. Even the oldest Linux system command dd cannot handle the transfer properly. Perhaps the current state of the driver is incompatible with some component of the system, for example, with the new version of *Fedora 39*, and XDMA simply does not deliver full performance due to some bug.
2. The Hardware Numbers program *(Fig. 7)* shows excellent results. This Xilinx program was created so that we could learn the potential performance figures of our PCIe interface without software and drivers. Thus, the problem is definitely not in the hardware but somewhere in the OS or **XDMA**.
3. When reconfiguring the XDMA IP core from Gen 3x16 to Gen 3x8, I expected to see the same 5-6 GB/s as in Gen 3x16, but I saw 2.2 GB/s. Both values for both configurations are ~30% of the maximum. Something during operation cuts the speed by ~70%.
4. I noticed that speed increases when I build and do insmod xdma on different versions of the Linux kernel. On 6.5.2, it works *10-20%* faster than on 6.9.9.
5. Different versions of the dma_ip_drivers repository do not yield significant results.

I hope that I can achieve better results on another computer. On the Figure 6 you can see my honest performance numbers. In Figure 8, you can see the test bench with the L2 hub.

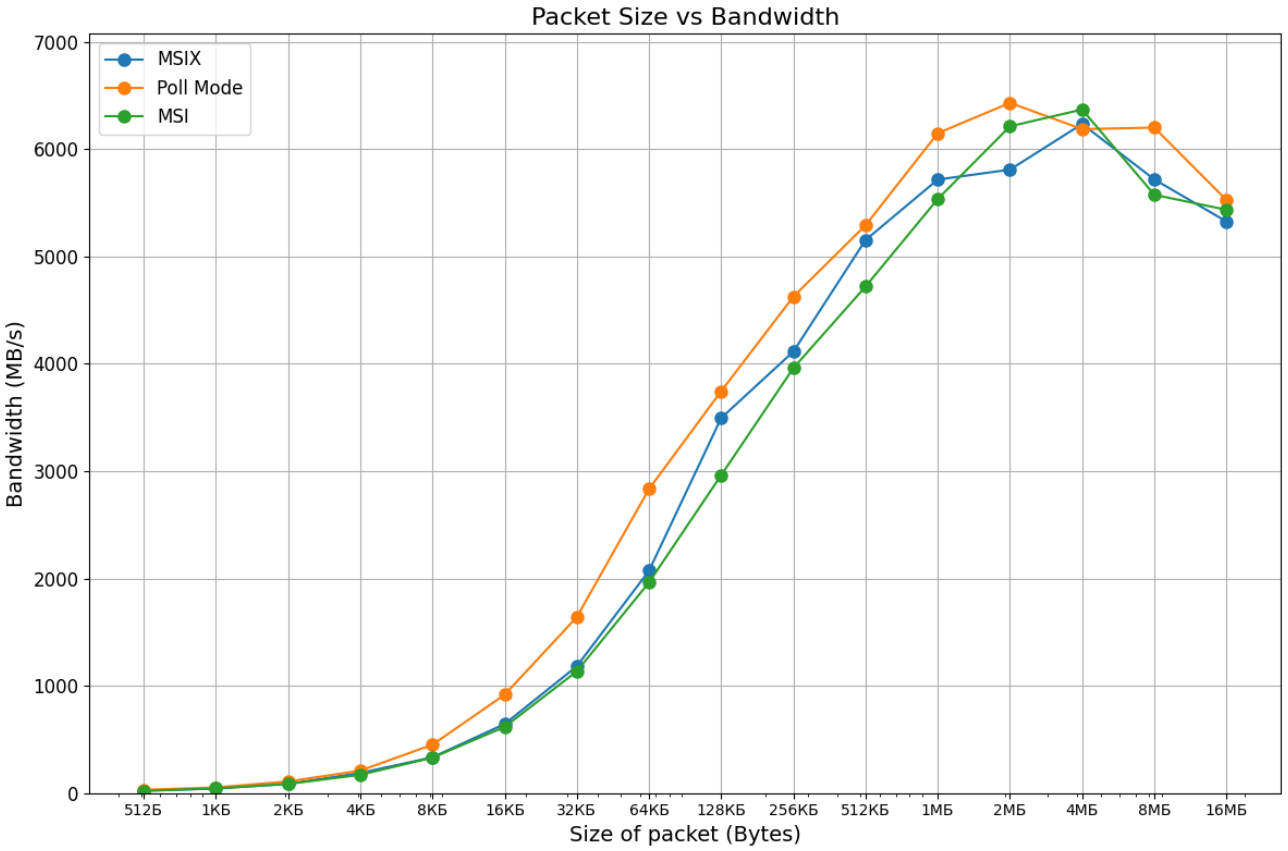

Fig. 6 Measure of performance with XDMA and Software

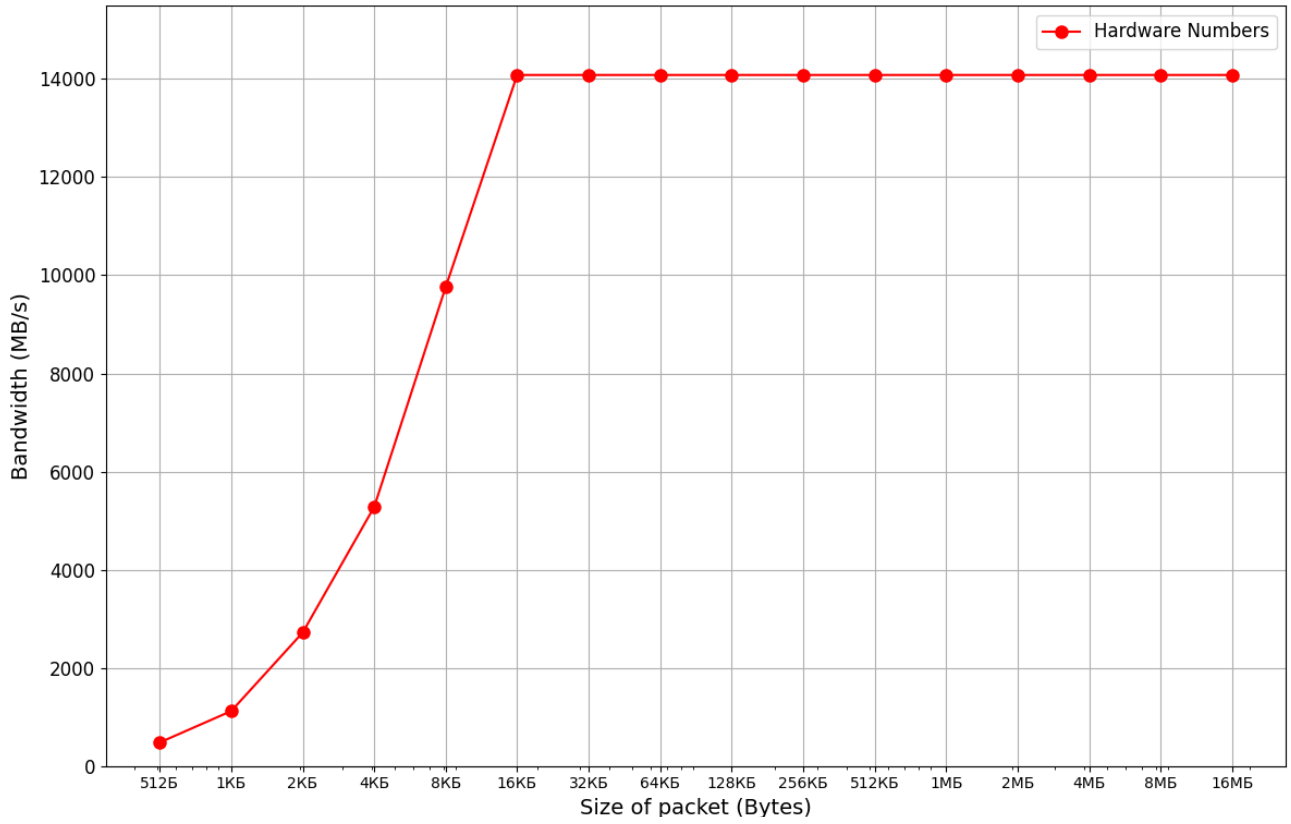

Fig. 7 Measure of performance without XDMA and Software (potential numbers)

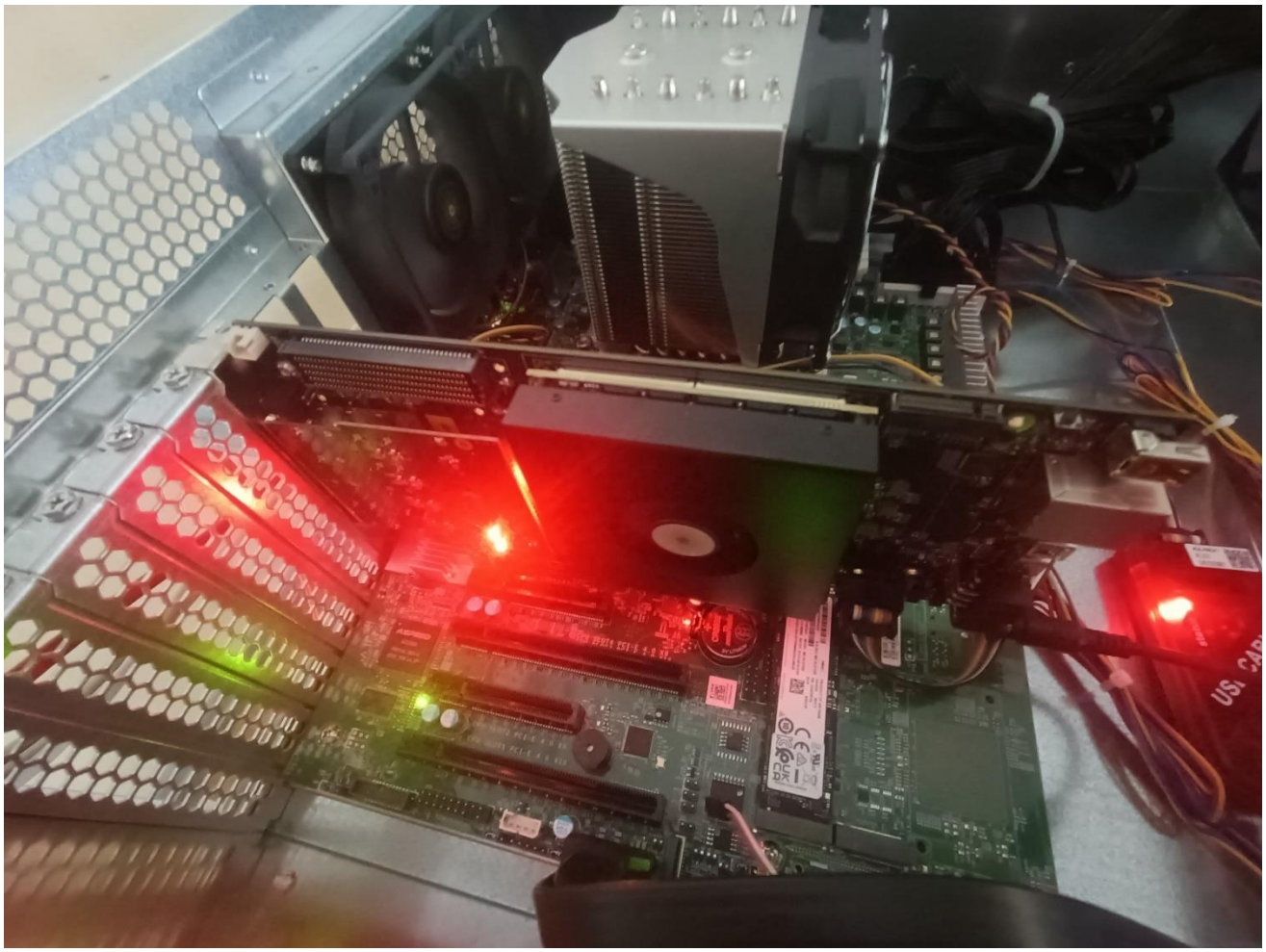

Fig. 8 Test bench with L2 hub

In the final work, the following laboratory work was conducted:

Our colleagues working on the L1 hub developed a packet generator for the L1. After that, they brought the L1 to our laboratory and connected it to our board (Figures 9 and 10). Then, we read the packet again on the computer (Figure 11).

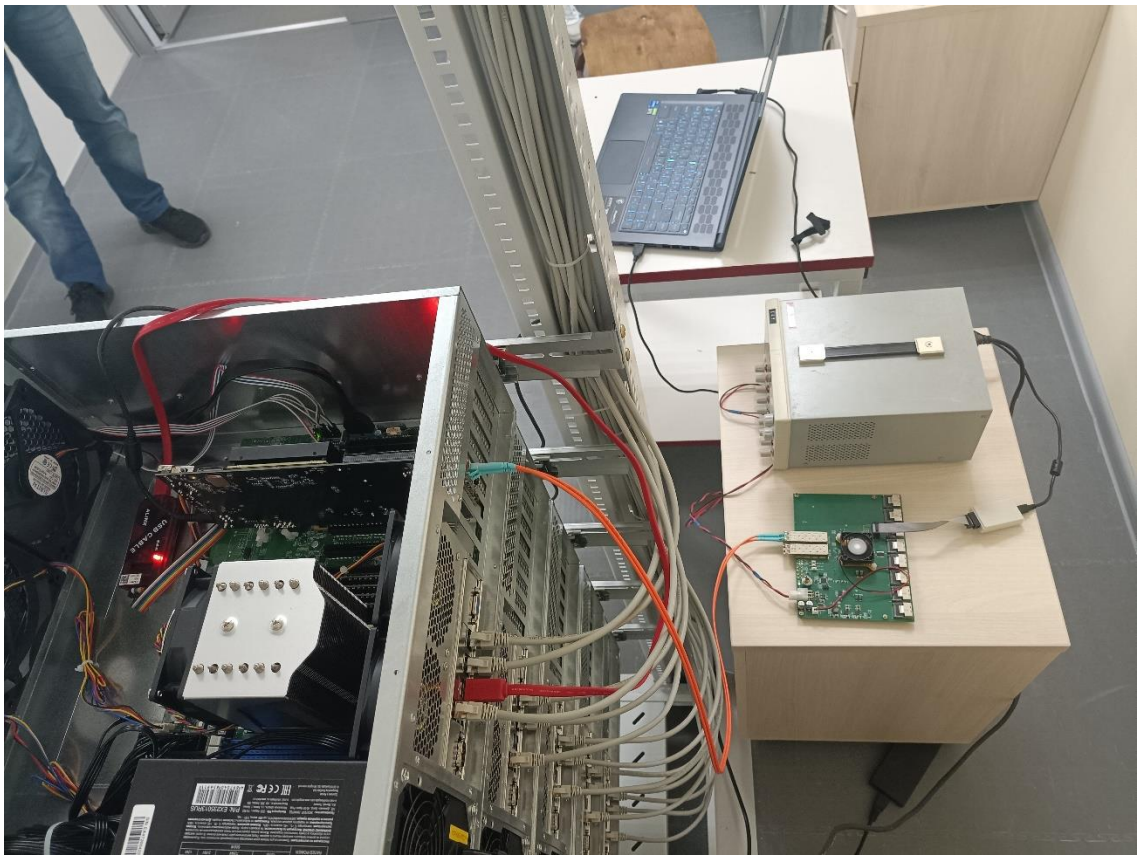

Fig. 9 Test bench with L1 and L2 hub

Fig. 10 Test bench with L1 and L2 hub

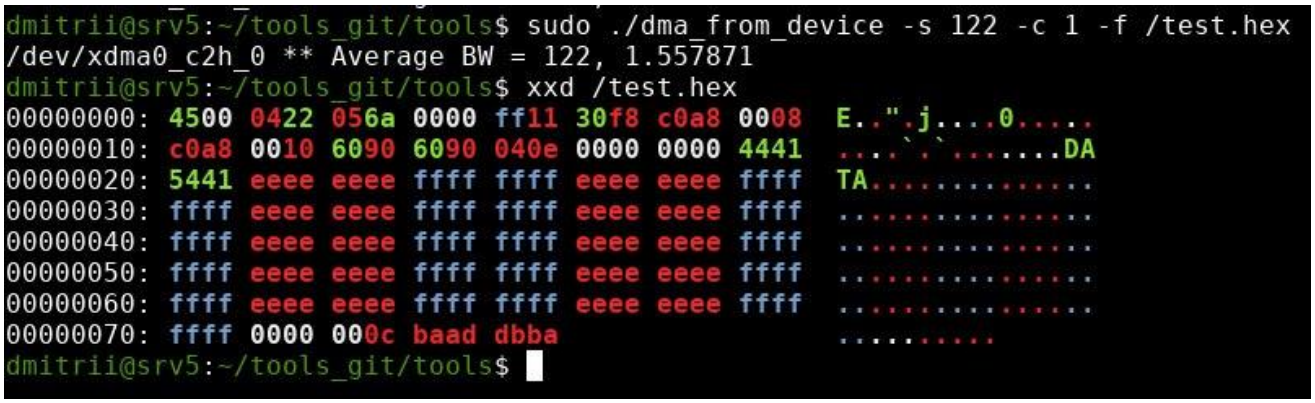

Fig. 11 L1 Packet on the computer

Conclusion

In summary, during my 6 weeks in Dubna, I successfully completed all three planned tasks. Throughout the process, I gained invaluable experience in computer setup and PCIe debugging. We confirmed that the technical aspects of the optics and boards are in order, and we can now confidently verify our design in the simulator.

Additionally, we improved our design by switching to higher data transmission speeds. I measured the speed and found that starting from a packet size of about 1 megabyte, the speed stops increasing. It is likely that this system is not compatible with the driver, so I will need to test the maximum speed on another computer. I plan to address this in September.

References

[1] http://nica.jinr.ru/

[2] https://spd.jinr.ru/

[3] Github XDMA https://github.com/Xilinx/dma_ip_drivers

[4] DMA/Bridge Subsystem for PCI Express Product Guide (PG195) https://docs.amd.com/r/en-US/pg195-pcie-dma/Introduction

[5] XDMA Perfomance Debug https://xilinx.github.io/pcie-debug-kmap/pciedebug/build/html/docs/DMA_Bridge_Subsystem_for_PCI_Express_XDMA_IP_Driver/performance_debug_checklist.html

[6] Perfomance Numbers Xilinx https://support.xilinx.com/s/article/68049?language=en_US

[7] UltraScale+ Devices Integrated Block for PCI Express v1.3 https://docs.amd.com/viewer/book-attachment/tCJDwogjyJ9_CE2Q_px23A/B6WE5d4mRhWPAlUpzHxkcw