Readout with DMA technique

Radosław Trębacz

Contents

Chapter 1

Data Aquisition System

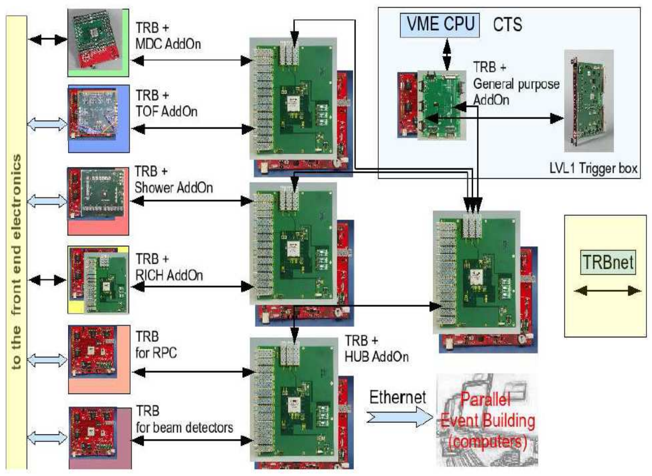


Figure 1.1: The new DAQ system.

Hades will be moved to the upcoming FAIR accelerator complex, where its experimental program will be continue up to (tutaj bym dal 8 GeV of kinetic beam energies per nucleon) kinetic beam energies per nucleon of 8 GeV. The average amount of data is expected to be $300MB/s$ and mean trigger frequency will be 20 kHz (in peak 100kHz). In order to take the data in such conditions the HADES detector, readout and trigger system is currently under upgrade.

An overview of the new Trigger and Data Aqquisition system is shown in Fig. 1.1. The complete readout chain consists of the following basic elements:

- front-end electronics,
- readout electronics,
- slow control and trigger distribution system,
- event building system.

The first level trigger (LVL1) decision is created based on the charged particle multiplicity taken from photomultipliers of the Tof and Tofino detectors. It arrives in very short time $t < 100ns$. If LVL1 is positive, the next data from the RICH, TOF and a Shower detectors is taken into account to calculate the LVL2 trigger decision. This algorithm is optimized for searching electron pairs candidates. The positive decision of LVL2 induces a full readout of data from all sub-detectors. After data has been read out from the front-end electronics, a level-2 (LVL2) trigger algorithm selects events by searching for electron candidates. In order to handle a latency which coresponds to several events (on the average it is 5-10 events), the readout boards must have buffers large enough to hold the data for this time. If a LVL2 decision is issued, the data of the corresponding event is send (UDP network protocol is used for this purpose) to the Event Builder (EB). The EB is a PC which combines the data from different asynchronous data sources into complete events and finally writes them to mass storage.

Up to now each of subdetector owns individual system to readout data from it. In the new design the "Trigger and Readout Board"(TRB), shown in Fig. 1.2, will be used as a general platform for all subsystems. Together with a given AddOn board, that is assigned to particular subdetector, TRB composes one physical unit.
The current version of TRB contains:

- Etrax-FS processor with $128MB$ memory connected to the 100 MBit/s Ethernet, where a standard Linux 2.6 kernel is running. It will be decribed later,

- 128-channel Time to Digital Converter electronics (tutaj bym dal ASICs), with time resolution $\sigma = 40ps$, based on the HPTDC from Cern,
- optical link with throughput of $2GBit/s$,
- programmable logic FPGA (Xilinx Virtex 4) connected to all main components on the TRB to manage data flow on the board,
- TigerSharc DSP,
- a high data rate digital interface connector (32 LVDS lines, 15GBit/s and 32 TTL lines). It gives the possibility to mount AddOn boards to the TRB which provide the detector specific interfaces or additional computing resources.

All the detector specific functions are performed by detector dependent AddOn boards which is mounted on the TRB. According to this concept, following AddOn boards were built:

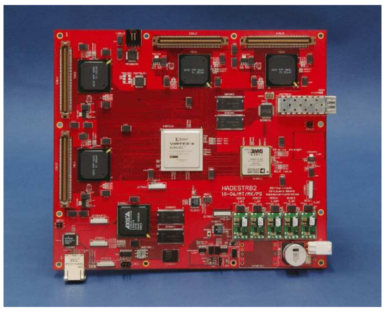


Figure 1.2: The front view of the Trigger and Readout Board.

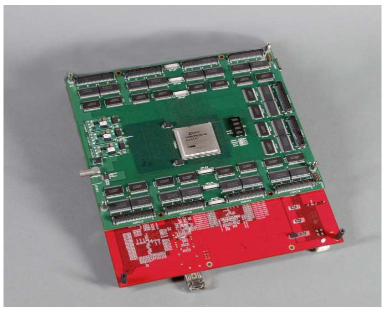


Figure 1.3: The MDC AddOn mounted on the TRBv2 top. Here, the back side of the TRBv2 is visible in part.

1.0.1 The device driver for Etrax FS and Direct Memory Access technology

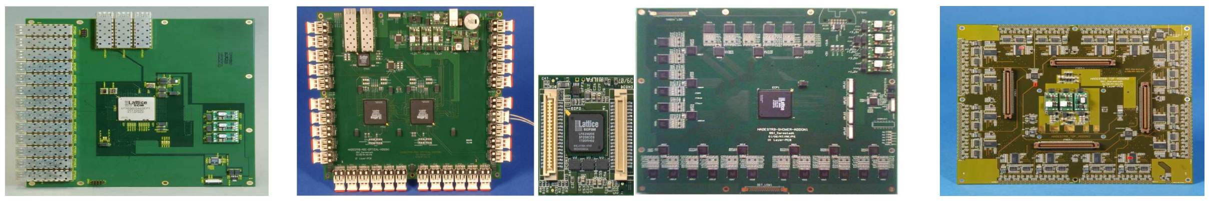
It is important to mention that in this chapter the second version of TRB is presented. A first version (TRBv1) was used successful in the beam time April 2007 to read out the forward wall and the beam detectors.

The TRBv1 was fully integrated into the Hades DAQ system. 80 kHz on LVL1 (with large down scaling) was achieved and LVL2 rates up to 18 kHz, which corresponds to data rates of 1.8 MB/s. One should point out, that without DMA capabilities on the Etrax, the processors performance has to be shared between the Hades sub event building software and the readout from the LVL2 memory.

The design goal after optimization is 80kHz LVL1 rate and a LVL2 bandwidth of 10 MByte/s. In order to make it possible we decided to use direct memory access (DMA) technology in the process of readout. To explain it, first the description of Etrax FS, fig.1.4, in detail is needed.

The main elements of EtraxFS chip are:

- 200 MHz RISC CPU with a 32-bit data and address width, where the standard Linux kernel is running,
- micro-code programmable I/O processor consisting of three 200 MHz 32-bit processors with local memory and hardware accelerators for real-time performance,
- 10 DMA channels each with 64 bytes FIFO,
- dual 10/100 Mbit/s full duplex Ethernet,



for: Rich, MDC, Shower and Tof.

- several synchronous and asynchronous I/O ports with 80 read/write configurable I/O pins

As it can be seen in figure 1.5, the I/O Processor is not a processor in a conventional sense. From a software perspective it should be considered as a collection of blocks, which are connected to each other in a chip-specific manner. However, the flow of data is configurable with regards to which hardware blocks the data should traverse on its way to or from a peripheral device.

The I/O processor contains one Master(MPU) and two Slaves(SPU) Processing Units. When the first one can be consider as a traditional processor with interrupt(IRQ) handling capabilities, SPUs differ from an ordinary CPU, because they can't handle interrupts and SPUs can execute state-machine code in a special mode called FSM-mode. The one of the most importent module presented in Fig. 1.5 is the switch, which is used to configure the individual connections between the modules of the I/O Processor.

The connections are defined by registers inside the Switch in the way to construct a chain, where data will be transfer from I/O ports to DMA channel. After that data are (is

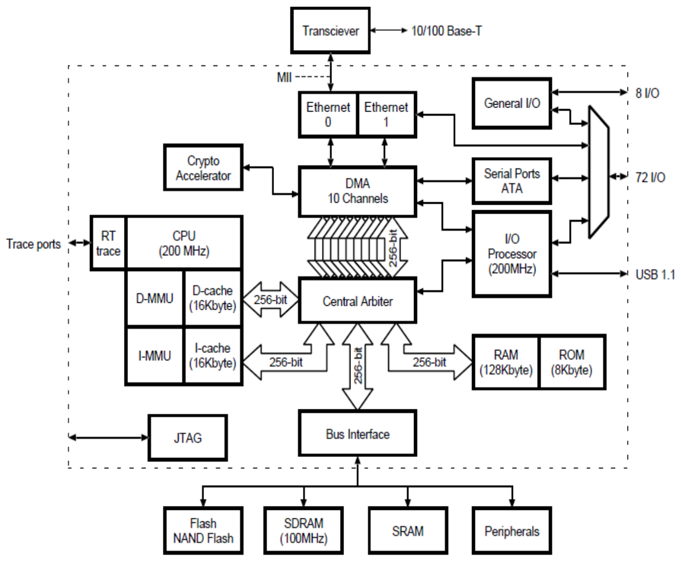


Figure 1.4: Overview of the Axis Etrax FS.

?) handled by a device driver.The DMA channel is a bus using DMA technology(to zdanie troche takie maslo maslane). It means that moving(reading/writing) data inside this bus happans without using the central processing unit (CPU). The wholeness was presented in the Fig. 1.6.

Regardless of used AddOn boards (or in the case when front-end electronics are connected directly to TRB) experimental data (goes ?) to FPGA chip. In the case of positive LVL2 trigger decision it is forwarded to Etrax FS via I/O ports and the parallel data path(PDP). After dma trigger sent by SPU0, data as events (collection of words) is stored by device driver in the dedicated buffer. On demand of readout application these events are copied from this buffer and sent via ethernet to the Event Builders all the time. This visual draft was presented in the Fig.1.6 inside violet box.

The protocol between FPGA and Etrax FS is presented in the same figure, but in the yellow frame. When 32-bits word was written on port B and C, a trigger from FPGA on the pin 16 of the port B is set. Then data are automatically readout by Etrax chip and it is routed over a dedicated path to DMA Communicator-In(DMC-In). Because of the

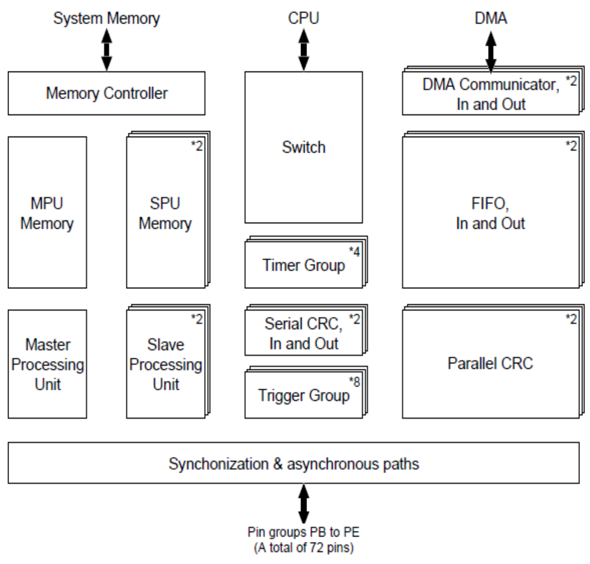


Figure 1.5: Block diagram over the I/O Processor.

size of DMC-In buffer FPGA sends data to Etrax as a chain of 15 words with a constant frequency. SPU was set to be sensitive for trigger going from FPGA and after the first-in-

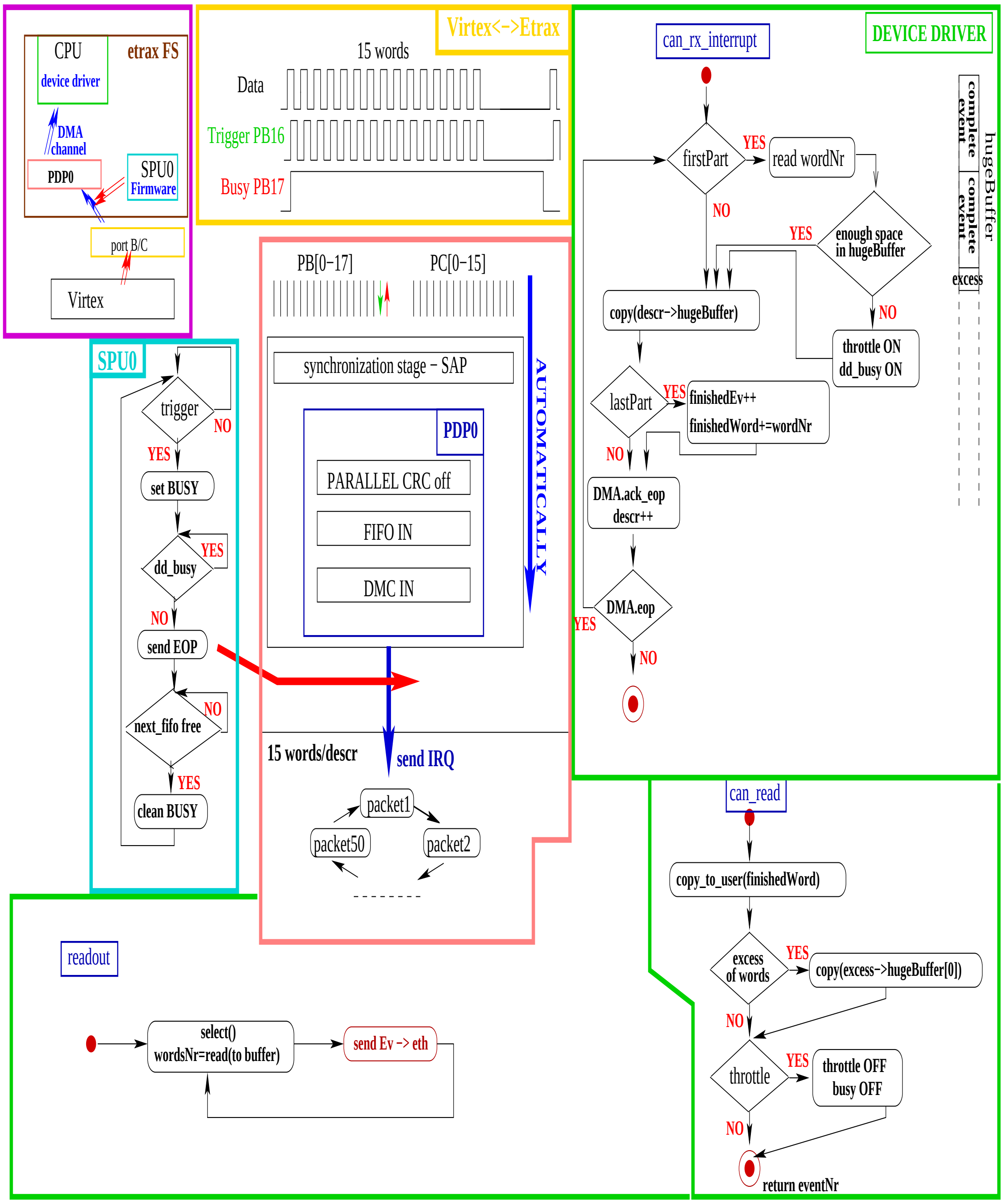


Figure 1.6: The data flow diagram.

the-chain FPGA trigger SPU sets busy signal on pin 17 of the port B. This signal informs FPGA to not send next chain of words. In this state FPGA chip can only finish writing a current chain of data. After the busy state was set, SPU is waiting some time needed to gather 15 words and then it checks if a device driver busy state (”dd_busy”) is set. If not, SPU writes 16. word to DMC-In which contains EOP(end of packet) signal which induces rewriting data from DMC-In buffer to DMA channel. It means, in practice, that data are available by the device driver. This is the main step of SPU logic. When this is done, the busy signal is clear and the whole cycle is repeated (Fig. 1.6, blue box).

The data path inside modules managed by the switch is presented in the Fig. 1.6, pink frame. In the previous paragraph the automatically readout of data was mention. The logic values from pins on ports B and C are read out and synchronized by the Synchronization and Asynchronous Paths (SAP) module. In the moment of trigger arrival the Synchronization and Asynchronous Paths (SAP) module readout the logic values from pins on ports B and C. Data is synchronized (32-bits word as BUS1 is created and from this moment) and adapted to the I/O Processor 200 MHz clock. After that the word is going to the Parallel Data Path In (PDP-In) module. It consists of Parallel CRC, Fifo-in and DMA Communicator In (DMC-In) modules. Each of these modules are connected with the next one and data cross freely them up to DMC-In. When the word with EOP sending by SPU reached DMC-In, an eop-interrupt to CPU is sent. The next words will be written to the next packet. These packects are, we can say, interface between SPU and CPU. SPU controls writing data to them, while CPU is reading them and sending to readout application. There is no need to sending any information about availability of empty packets, because this is control by SPU (”next_fifo free” request).

The eop-interrupt calls ”can_rx_interrupt” function in the device driver, Fig. 1.6, one of the green frames. This part of driver logic handles the packets. Most of the non-empty events contain more than 15 words, therefore usually more than one packet belongs to one event. Taking it into consideration one needs to read a header of each event from the first packet of event and extract a number of words in a given event. Then the packet is copied (always 15 words) to a main buffer and if the last packet of event arrived the position of the next event in the main buffer will be adjust to the real size of previous event. To take into consideration that data in the data aquisition system are 64-bit aligned, when the number of words in the event is uneven, one empty words is added. If the main buffer has no space for the next event, "dd_busy" (and throttle variable) is set.

The function, described above, responds for the request from SPU and is responsible for the addition of events to the main buffer. On the other hand, there is "can_read" function. It is triggered by "readout" application and copies a finished events to its. It could happen that "can_read" function was called, when the main buffer contains not only the finished events, but additionally one partly-completed event. In this case, this unfinished event is copied at the beginning of main buffer. If "throttle" was previously set to 1, after push the content of main buffer to "readout" it is reset, and also "dd_busy" is set to 0.

The last element on the way of event from I/O ports to ethernet is the user space application "readout". It consists mainly of three methods:

- select(), to wait if at least one event is located in the main buffer
- read(), to read finished events to the local buffer
- NetTrans_send(), to send data through the ethernet to the eventbuilder

After implementation of the above logic the rate of events increased from $40k/s$ to $125k/s$ with covering 15 words per event, which corresponds to $7.5MB/s$ of UDP ethernet transfer.