THE READOUT BUS OF THE ATLAS LEVEL-1 CALORIMETER TRIGGER PRE-PROCESSOR

C. Schumacher

Institut für Hochenergiephysik, Heidelberg, Germany
(e-mail: schumacher@asic.uni-heidelberg.de)

representing the ATLAS level-1 calorimeter trigger group

Abstract

The input to the ATLAS level-1 calorimeter trigger consists of 7200 trigger tower channels. The Pre-Processor system of the Level-1 calorimeter trigger provides facilities to read out this raw data on which the trigger decision is based. A high-bandwidth custom bus system, built from parallel point-to-point links, is used to collect data from several VME modules. These data are fed to the S-Link transmitter used for sending the read out data to the ATLAS DAQ system. The architecture of this bus system and results of first tests, performed within a modular test system, are presented.

1 The Pre-Processor system

All trigger data coming from the ATLAS calorimeters have to pass the level-1 trigger Pre-Processor system before it can be processed by the trigger. The Pre-Processor system performs digitisation of analogue input data, bunch-crossing identification and energy-calibration and then sends the data to the succeeding level-1 trigger processors. These generate an accept signal, which selects events for readout and further processing. Figure 1 shows a diagram of the Pre-Processor system.

An important aspect of the Pre-Processor system is the readout of raw trigger data. This is required to monitor the function of the trigger and to verify that the trigger data and the data from the seperated path of full-granularity readout of the calorimeters are consistent.

The Pre-Processor system consists of 128 VME modules, the *Pre-Processor Modules* (PPM), which process 7200 channels of trigger input data. Most of the processing is performed by an ASIC, the *Pre-Processor ASIC* (PPrAsic). The readout to the standard ATLAS data aquisition (DAQ) system is performed by 16 Readout Drivers (ROD), which format the read out data and feed them to the standard ATLAS readout link, the *S-Link* [1].

The collection of readout data is done in two steps. First it is collected on board-level by the so-called *Readout Merger ASIC* (RemAsic) using serial links. to the PPrAsics. In the second step data are collected from several modules and sent to the ROD, which then transmits it to the DAQ. For the inter-module collection of data a custom bus system is used. It is built from pipeline elements, which are connected by point-to-point links in a ring-like fashion. This bus, called the *PipelineBus*, is presented in the following sections in more detail.

The PipelineBus can not only be used for readout but also for sending configuration data to the connected modules. The ROD takes the role as source of these data and sends them to all connected Pre-Processor modules.

2 PipelineBus structure

A PipelineBus ring consists of several bus nodes. For the Pre-Processor system three different kinds of nodes are used. One ring consists of a *master node*, a *S-Link node* and several *readout nodes*. Figure 2 shows this configuration. The nodes and their connections form a closed pipeline.

The master node controls the bus by putting control commands into the pipeline. The commands propagate through the pipeline from one node to the other and are finally received back by the master. Because of the ring structure of the bus the master is able to check the reactions of the other nodes, which resulted in new or modified data on the bus. It then can take
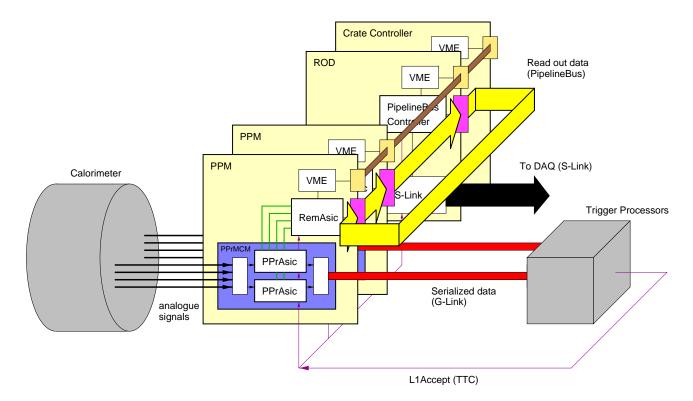
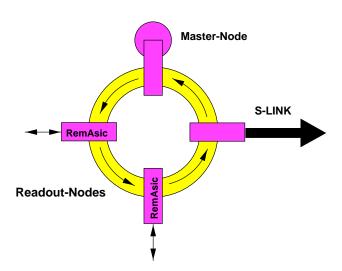Figure 1: The ATLAS level-1 calorimeter trigger pre-processor system

appropriate actions like starting or stopping readout operations or initiating error recovery procedures.

Each Pre-Processor module acts as a readout node for the PipelineBus. On request of the master it puts event data on the bus. The data then propagate along the other nodes until it reaches the S-Link node.

The S-Link node takes the event data, which it receives from the bus, formats it to the standard ATLAS event format [3] and sends it to the DAQ system using the S-Link transmitter.

The I/O part of all nodes is contructed in the same way as shown in figure 3. It consists of a 35 bit wide register and a multiplexer. The latched input data are available to the node for further processing. By using the select line of the multiplexer the node can control whether it just passes on the received data to the next node or if it injects new or modified bus data. All nodes are controlled by a common clock signal. Therefore the bus could also be seen as a kind of parallel shift register or FIFO.

With each clock tick the bus data are moved from one node to the next. The data words propagate through the bus in a pipelined way. With a clock frequency of 40 MHz the bandwidth of the bus amounts to 166 MByte/s. This includes protocol information.
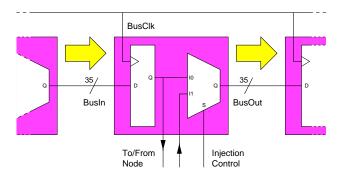

Figure 2: The topology of a PipelineBus as used in the Pre-Processor system

Figure 3: Structure of a PipelineBus node

The connections of the bus nodes are done by point-to-point links of a width of 35 bit. The advantage of point-to-point links is, that they are electrically and mechanically simple. That allows fast and reliable signal transmission. Also there are no big driver strengths necessary. Signals can be transmitted from chip to chip without additional buffers.

3   PipelineBus protocol

The 35 bus lines include a parity bit for error checking and two control bits, which identify the type of the bus word consisting of the remaining 32 bits of data. There are three different types of bus words, *empty slot*, *command* and *data*. The fourth control bit combination of 'b11 is unused and considered an illegal bus state.

Empty slot is the default state of the bus and means that the bus is unoccupied. Empty slots can be used to put commands or data in without restrictions. The contents of the 32 bit bus word is arbitrary.

Commands are used to control the bus. They are injected by the master node into the pipeline and processed by all nodes that the command addresses. Therefore the command word contains a 6 bit *address* field. An 8 bit *token* field is used to identify the command. 16 bit of *argument* data are available for commands which require parameters. The same field is used to store return values, which are generated by nodes as response to a command.

Two additional one-bit fields are available: an *accept bit*, which is set, when a node has processed the command, and an *error bit*, which indicates that an error occured. If a bus node detects an error, e.g. a parity error, this bit is set and an error code is put into the argument field. The bus master has to process these error words and take appropriate actions.
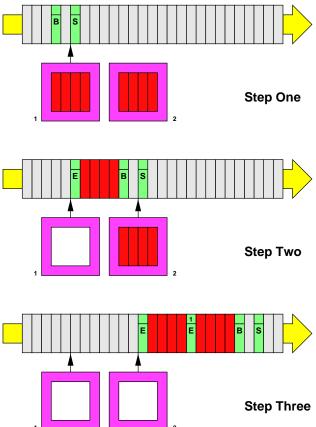

Figure 4: A PipelineBus readout operation

There exist 15 different commands. There are commands for address configuration, control of readout and input, status information and commands, which are used to delimit data blocks.

If the control bits indicate that the type of the bus word is data, all 32 bits are used for user data.

As an example for a PipelineBus operation the process of reading out a few data words is depicted in figure 4. It is shown in three steps each representing several bus clock cycles. The big horizontal arrow stands for the pipeline and indicates the direction the bus words propagate. The two boxes below represent two readout nodes.

In step one the bus master has injected the command `StartReadout`. It propagates through the pipeline and when it passes the first readout node, the node enters readout mode, which is indicated by the small arrow above the node. In readout mode the node waits for the command `BeginOfData`, which activates the actual readout.

When the node receives the `BeginOfData` command

it looks for empty slots following the command and fills them with data. This is shown in step two. When a complete data block is written to the pipeline it is terminated by the node by an `EndOfData` command. The argument of this command contains the number of the node the data block belongs to. Then the node waits for the next `BeginOfData` command to insert the next block of data.

In step three the `BeginOfData` command has passed the second node. The node waits for the next empty slots and lets pass the data block of the first node. Then it fills the empty slots with its own data block and terminates it with an *EndOfData* command. After that the readout operation is finished. The event block, which was built during this opeation, is now propagating through the pipeline to the S-Link node, which adds header and trailer information and sends it to the DAQ system.

The structure of the PipelineBus fits well to the S-Link interface. So only a small amount of formatting is necessary to build standard event fragments.

4   Test system

For testing the PipelineBus and other components of the Pre-Processor a flexible test and development system was built. It is based on a general-purpose motherboard, which contains commonly used functionality. Special functionality is added by application-specific daughterboards.

The motherboard (see figure 5) is implemented as 6U VME module and provides two CMC slots for inserting daughterboards. In addition to standard CMC cards the slots can also be used for S-Link cards.

Circuitry on the motherboard includes 32 kByte of RAM, a FPGA and a clock generator. It also provides the generation of a 3.3 V supply voltage.

Two PipelineBus daughtercards are used in the test system. One card represents a readout node and uses a prototype of the RemAsic [2] (see figure 6). The other card represents a master node. The logic for the master is implemented in the motherboard FPGA.

CompactPCI connectors are used on the front panel for connecting the bus nodes. The actual connection is done by small printed circuit boards, which connect two adjacent modules. To close the PipelineBus ring a flat ribbon cable is used. The middle row of the connectors is used for common control signals like the bus clock or the level-1 accept signal. Figure 7 shows a motherboard equipped with a master CMC and an I/O control card, which supplies these control signals.
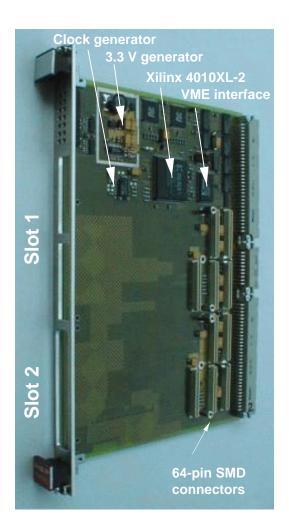

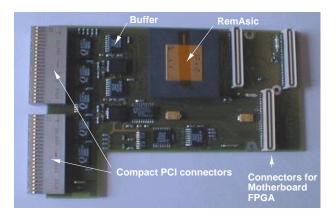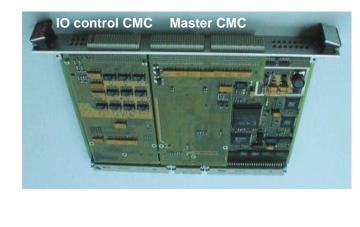Figure 5: The motherboard of the Pre-Processor test system

Figure 6: The RemAsic daughtercard used in the Pre-Processor test system

Figure 7: An assembled master module and the front-panel view of the PipelineBus test setup.

It also shows a view of the front panels of two modules and the location of the PipelineBus connections and the control signals.

5 Software

For the Pre-Processor test system a software package was written, which provides access to all hardware components, like registers, memories or FPGAs (see figure 8). By the use of text configuration files for register definitions etc. in many cases no programming skills are required to integrate new or modified hardware into the software framework. A VME bus client/server module allows to access hardware by a network connection.

The software is based on an object-oriented design and was implemented with C++. For the graphical user interface the toolkit Qt by Troll Tech [4] was used. The software was tested on different platforms like Solaris, Linux, HP-UX and LynxOS and could be ported to Windows without a major problem.
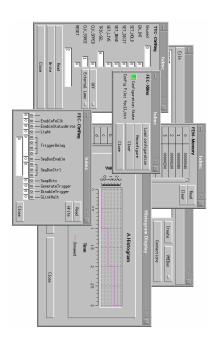

Figure 8: Screenshot of some of the GUI controls available in the Monitoring and Control Software

6 Conclusions

The ATLAS Level-1 Calorimeter Trigger Pre-Processor system provides facilities for readout of raw trigger data, which is important for monitoring and verification of trigger operation. A custom bus system, the PipelineBus, was developed to collect data from several VME modules. It provides high bandwidth with relatively low complexity. A flexible test system was developed based on VME and CMC standards. This includes control and monitoring software. It has already shown to be a useful tool for hardware tests.

The design of the PipelineBus, the system and software is not limited to the Pre-Processor system. Other systems could also benefit from this developments.

References

[1] O. Boyle et al., *The S-LINK Interface Specification*, ECP-Division CERN, Geneva, Switzerland, 27 March 1997,

[2] *Remasic User and Reference Manual*, http://wwwasic.ihep.uni-heidelberg.de/atlas/docs/remman

[3] C. Bee et al., *The event format in the ATLAS DAQ/EF prototype -1*, ATLAS internal note, ATL-DAQ-98-129, 27 Oct 1998

[4] Troll Tech, http://www.troll.no