The Readout Driver for the ATLAS Muon End-cap Trigger and its architecture and design language techniques

D. Lellouch, L. Levinson, A. Roich

Weizmann Institute of Science, Rehovot, Israel 76100
Lorne.Levinson@weizmann.ac.il

Abstract

The prototype of the Read Out Driver for the ATLAS Muon Endcap trigger system is described. The hardware is based on a single large Xilinx Virtex FPGA that accepts data from four gigabit optical links and sends processed output to the ATLAS central DAQ via an S-link and sampled data to a VME processor via the VMEbus. A C-like procedural language was used for a large part of the design. In addition to using pipelined logic and other standard FPGA design techniques, the design uses some architectural elements that are more common in microprocessor architectures. The ROD's design features, implementation details, DAQ software, experience with the procedural language and performance are described.

I. Overview of the Readout System

The ATLAS Muon Endcap trigger has a hierarchical readout system for 320,000 binary channels from Thin Gap Chambers, "TGCs", see Figure 1. The "Read Out Driver", ROD module, one for each octant, collects data via 13 optical links, sends an assembled event via an output S-link to the ATLAS central DAQ, and then sends a small sample of the event data via the VMEbus to a commercial VME processor. Each octant consists of ~180 on-chamber trigger-readout ASICs whose data is concentrated by 12 Star Switches. Each Star Switch, i.e. ROD input link, provides data from 700 to 1900 binary channels. An important feature is that the Star Switches do a partial sparse data scan and send the ROD partly zero-suppressed data. Consequently the ROD must handle variable length records. The estimated input bandwidth per octant (ROD) is 22MB/sec, with no safety factor. The data is read out in response to a Level-1 trigger, which can occur at a rate up to 100kHz. Table 1 summarizes the major ROD requirements.

A. Character of the input data

The data is dominated by cavern background: about 10 uncorrelated hits, i.e. from neutrons and gammas, and about 1.5 correlated hits, i.e. from charged particles, per Level-1 trigger per octant. The charged particles result in hits is several layers of chambers. The simulations [1] of the cavern background giving these occupancies have a large uncertainty and therefore a safety factor of 10 should be applied, especially since the ROD, sitting at the top of the hierarchy, is the bottleneck for the TGC readout.

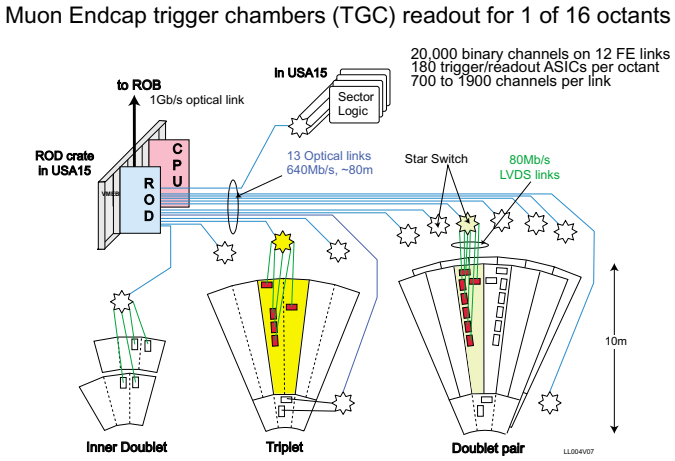


Figure 1: Readout of one Muon Endcap trigger octant.

Due to the low occupancy, the data is dominated by headers. These are still required to confirm that all the bunch crossing and Level-1 ID counters in the on-chamber ASICs are synchronized.

In addition to chamber hits, the data include the output from the coincidence logic of the on-chamber ASICs. These are 2-out-of-3 or 3-out-of-4 coincidences, called "tracklets" caused by the correlated hits of charged particles traversing 3 or 4 chamber layers. These provide a monitor of the proper functioning of the on-chamber coincidence logic, which is crucial to the integrity of the muon trigger.

Table 1: Required functions of the ROD

Receive TTC information and queue expected event Level-1 Accept IDs.
Collect event fragments from several Front End links corresponding to each of the Level-1 Accept Event Ids.
Detect and recover from input link errors and data errors.
Assert RODBUSY to the ATLAS CTP module when necessary, but as infrequently as possible.
Extract hit and trigger data from zero-suppressed bit map
Provide sampled full events, hits and tracklets to the ROD Crate Processor for monitoring the system.
Format events into ATLAS standard ROB format.
Send the data to the ROB and/or Rod Crate Processor.
Respond to flow control signals from the ROB.
Other requirements for calibration and monitoring.

II. ROD Hardware Implementation

Figure 2 shows a block diagram of the implementation of the TGC ROD prototype. It is based on a Xilinx Virtex FPGA with 10800 logic cells (flip-flops) and 140 blocks of ½KB width-configurable dual port RAM. The board is a 6U VME board with three programmable clocks (1 to 100MHz) and 1MB of ZBT SRAM. A PLD provides various services. TTC signals can be received either as NIM or LVDS signals. The four input Front End links are implemented by opto-transceivers and Agilent G-link deserializers on four daughter boards. The output is via a standard S-link connector, on the back of the motherboard. The board is sufficiently general to allow its use also as a source of simulated data for driving Front End links to a ROD. For this use, the ZBT RAM is used as a store of simulated events. The daughter boards implementing the Front End links also have transmitters.

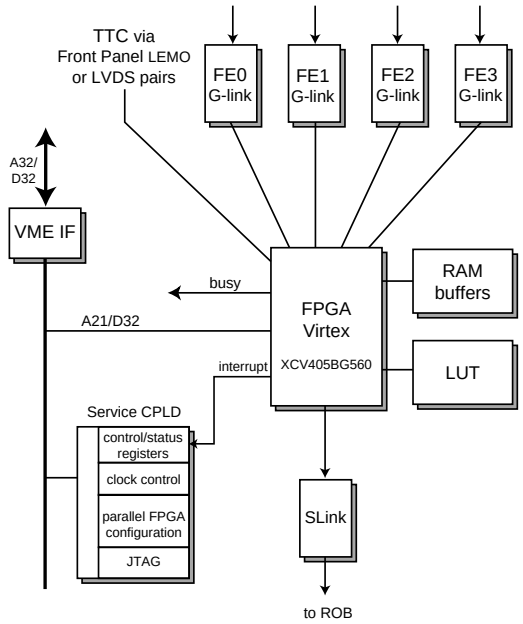
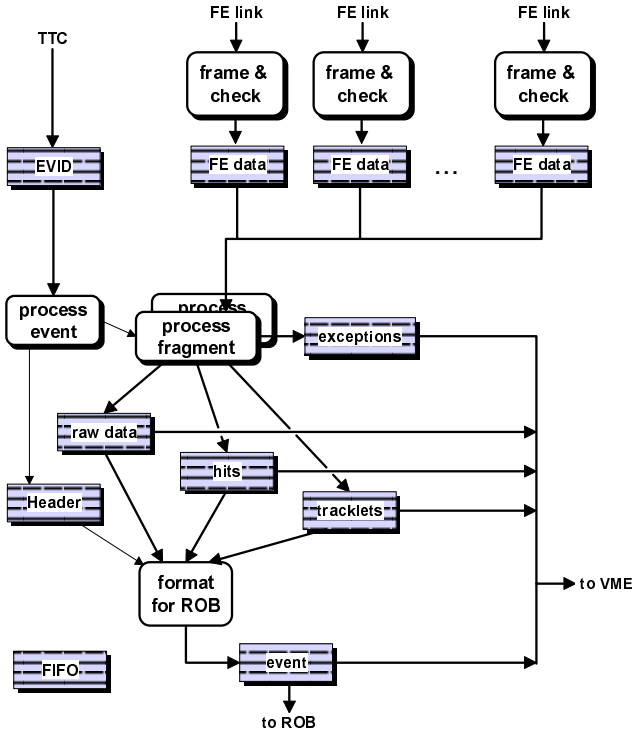


Figure 2: Block diagram of TGC prototype ROD

III. Data Processing

Figure 3 shows a simplified view of the data processing in the ROD FPGA. The design consists of a pipeline of variable latency processes connected by FIFOs. Unlike a registered pipeline, each pipeline stage, i.e. process, has fluctuations from event to event in its execution time and amount of output for the next stage. Each FIFO is a FIFO pair: one for the data words and another for a control word indicating the number of data words in a "record" and their status. The reader usually waits for an available item in the control FIFO, which is written after the data FIFO is written.

The first stage is a set of parallel processes that for each link checks each event fragment's framing, stores the fragment in the data FIFO and records the fragment's word count and any link errors in the control FIFO. The Level-1 trigger information is stored in another FIFO. The second stage is the main fragment processing: For each Level-1 trigger all the fragments are processed: their format is verified, Level-ID and Bunch Crossing IDs are matched, hits and tracklets are extracted by decoding the partially zero-suppressed bit maps and written to FIFOs. There is more than one fragment processor working in parallel. Additional output FIFOs from this stage sample hits and tracklets for the ROD crate processor monitor tasks to read via the VMEbus. The final stage formats the event in ATLAS standard ROB format and sends all



events out the S-Link. Events are also sampled here for the ROD crate processor.

Figure 3: Simplified view of data processing in the FPGA with internal FIFOs.

IV. Architectural Elements and the Hardware Description Language

The FPGA design was implemented using graphic tools that produce VHDL (Mentor Graphics HDL Designer), VHDL and Handel-C [3] with VHDL output. Mentor Graphics Leonardo synthesized it all for the Xilinx Place and Route tool. In addition to providing a high-level language to express the complex procedural processes, Handel-C enabled the easy incorporation of high-level architectural elements that simplify and modularise the design. After a brief summary of Handel-C, we discuss some of these elements.

A. Overview of Handel-C

The Handel-C language follows standard C syntax. Variables and arrays are implemented as registers or memory (either internal or external to the FPGA). Every assignment statement takes exactly one clock. In addition, several features from the Occam language developed for transputers have been adopted: Statement blocks can be identified as *sequential* or *parallel*. In the former all statements are executed one

after the other, in the later, in parallel (the block finishes when the last statement finishes). Parallel and sequential blocks can be nested in each other to any depth. Different sequential "threads" running in parallel can communicate and synchronize by means of passing single words through *channels*. These work as follows: when one thread issues a channel input (output) command it is suspended until the second thread executes the matching channel output (input); then, *in one clock*, the transfer takes place and both threads continue. It is easy to make a large number of variable latency parallel threads and synchronize them. Complex procedures making use of FPGA parallelism can be expressed. The design is synchronous by definition. Rich (and recursive) macros can be defined to create multiple instances of hardware structures. Despite the "C" syntax, the specific FPGA architecture, HW concurrency, how structures and algorithms synthesize, and timing constraints must be understood. Handel-C can produce VHDL or EDIF and can interface to VHDL and Xilinx cores.

B. Architectural elements

1) Embedded FIFOs

The large number of imbedded dual port memory blocks in the FPGA allows pipelines with variable latency stages to be constructed entirely within the FPGA. Embedded FIFOS are also used to bridge the several clock domains required by the ROD. Each Front End link process runs with a clock from its link's deserializer. In addition to the main design clock, there are separate clocks for the S-link and the TTC. The Front End link FIFO depths can be chosen according to the occupancy of the chambers they read out. The occupancies of all FIFOs can be monitored via the VME bus.

2) Thread-to-thread pipes

Thread-to-thread pipes are an extension of channels to a depth of more than one item. Almost all FIFOs in the design are implemented by a "pipe" object whose properties are: can read and/or write on every clock, read-on-empty and full-on-write conditions block (stall) the invoking thread until not-empty or not-full, can test empty, full and count anytime, can generate a Service Call to the host CPU on ~empty and almost-full, writes intended to execute on the same clock as the pipe transfer are guaranteed to do so. The last is important, for example:

```
par {pipe_write(hitpipe, hitid) ; hitid++ ;}
```

The incremented value of hitid must never be stored in the pipe, even if the pipe is full. This is done by stalling *after* the write if the pipe has just become full. (Note here how one instantiates a counter in Handel-C.)

3) Multiple "execution units" – fragment farm

The data volume varies from link to link and from event to event. There are 13 links but only about four fragment processors are needed to attain the required throughput. The processing capability should be matched to the average load, not to the number of links. Several fragment processors, FPs, are therefore instantiated as separate threads using the Handel-C "array of functions" construct. A dispatcher thread allocates each fragment to an FP on the fly using one channel per FP to pass the link ID of the fragment to process. The FP uses another channel to acknowledge completion. There is one output FIFO per FP. Another thread receives acknowledges and updates the list of free FPs. The size of the FPs justifies the additional multiplexing to implement this architecture.

4) Service calls

Several situations require sending a signal from a hardware thread to a software process on the host CPU. Examples are data ready for the host and exceptional conditions such as unrecoverable loss of event synchronization, link failure, and various time outs. Each of 255 different signals, i.e. SVCIDs, can be assigned for handling to a host CPU process, not necessarily to the same process. The scenario is as follows:

- An FPGA polling loop running over all conditions posts the SVCID to a VME register.
  - A channel is used to serialize posting.
- Interrupt host CPU.
- The host interrupt handler reads the SVCID and acknowledges it by clearing the SVCID register.
- The host interrupt handler sends a SW signal to the process that subscribed to this SVCID.
  - The host interrupt handler is now ready for another SVC.
- The host process performs the service, e.g. empties a FIFO.
- The host process writes a "done" acknowledge for this SVCID to the SVCACK register.
  - This SVCID can now be issued again.

5) FPGA to host CPU pipes

Some pipes have their read port on the VMEbus. Sampled events, hits and tracklets are passed to the monitoring tasks on the ROD crate CPU via such pipes. The host is interrupted by a Service Call either on a not-empty or almost-full condition. The SVCID indicates which pipe needs service. Particularly useful is an exception/message/debug info pipe. A message code, severity level and data bytes are written to this pipe and a software process counts, logs and perhaps services each exception. Hardware threads can dump values for debugging.

C. Advantages of Handel-C

We have drawn several conclusions on the suitability of Handel-C for the ROD design. Handel-C is most advantageous when the processing path depends on the data content, as opposed to, for example, matrix multiplication. The high level nature allows easy adaptation to demands for new and enhanced functionality. Handel excels in expressing complex arithmetic or logical algorithms and situations where there are many special cases, exceptions and error conditions. The high level does have a slight speed penalty, but the critical paths can be tuned or even done directly in VHDL. The advantages are less when the design centers on pipelines with fixed length data and deterministic latency and where there are only simple sequential data dependencies.

V. Integration and Performance

First, data with the Front End link protocol and format sent by the Star Switch was successfully received and basic ROD functionality verified. Then Star Switch emulators implemented by other ROD boards were used for further development, testing and integration. Several million event fragments simulated with the actual mix of Front End ASICs and occupancies from background were sent over Front End links to the ROD, processed, assembled into ATLAS standard format, and sent over an S-Link to an ATLAS DAQ PC running an event format checking program. Figure 4 shows the diagram of the integration test.

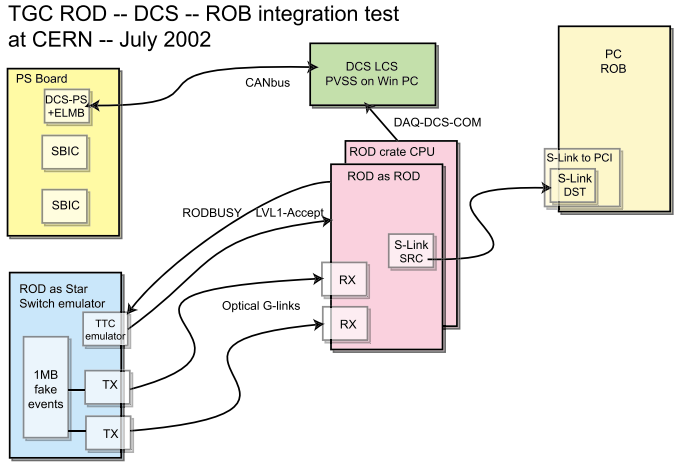


Figure 4: The integration test.

Online DAQ programs running in a ROD crate processor controlled configuration and data flow. This processor was a VME single board Pentium computer running Linux and many of the ATLAS Online software packages, including: Run Control, Information Server, Process Manager, Message Reporting System, the Configuration Database (simple use), VME driver and DAQ-to-DCS Communication (DDC).

At Start-of-Run, commands to configure on-chamber parameters such as chamber thresholds and ASIC parameters were sent via DDC to the DCS Local Control Station PC running the PVSS SCADA system. These were forwarded via CANbus to the on-chamber ELMB microprocessors for execution and status returned. In response to interrupts, sampled events and messages were read from the ROD via VME into a buffer manager [2] and checked for consistency by an intelligent event analyser and formatted dump. The Information Service provided a display of error counts and data flow statistics.

Measurements of maximum sustainable Level-1 event rate were made with sets of simulated events with occupancy varying from 0.1 to 20 times the expected: at expected occupancy the maximum Level-1 rate was 400kHz. At the maximum required ATLAS Level-1 rate of 100kHz the ROD was able to handle 20 times the expected occupancy. These rates are for one fragment processor handling the two worst-case Front End links in series. The final ROD will have four to six fragment processors for 13 links.

VI. Conclusions

- Using a large FPGA programmed with a high-level procedural language has delivered the required performance as well as the flexibility to respond to complex and advancing requirements.
- C-like high-level procedural languages allow for easy implementation and use of high level architectural elements such as multiple execution units, variable latency pipelines, multi-threading with thread-to-thread communication, and hardware to software messaging.
- C-like high-level procedural languages are a viable tool for real world FPGA designs, provided they are applied to suitable parts of the problem.

VII. References

1. Ian Dawson, FLUKA simulation of the ATLAS cavern background fluences, Configuration AV5, presented at the ATLAS Week, June 2000
2. KLOE Buffer Manager, Enrico Pasqualucci, INFN, Roma
3. Handel-C, Celoxica, Ltd., UK, http://www.celoxica.com