Error handling

In this note I try to summarise my understanding on the handling of errors, as we discussed during the collaboration meeting, October 97 (before I forget).

Assumptions on the architecture

In the rucksack there will be two or more ADC crates (VME, 9U?). Each crate will contain:

- A number of **ADC modules** (16?). Each module will serve a number of analogue lines (8?) which come from a HELIX token chain (8 chips for the MVD barrel). The subsection of the ADC module serving one chain is called **ADC unit.**
- An interface with the clock and control (**C+C interface**).
- A VME CPU connected to the network, which performs the readout and the management of the crate (downloading constants, handling unexpected conditions etc).

The system will be asynchronous, which means that each ADC unit will work independently.

Errors signalled by the ADC

Each ADC unit will be able to identify the following error conditions (see Katsuo's presentation on the WEB):

- Analogue data timing problems
  1. Time between trigger and data is wrong
  2. Time between two consecutive events is wrong
  3. Data length is wrong
- Pipeline out of phase

- <u>Analogue data timing problems:</u>

  1. The time between a GFLT trigger and the appearance analogue data (**DataLatency)** is incorrect (only timeout or precise clock counting?).

  From what I understand from HELIX manual, this DataLatency is (at HELIX level), 9 clocks + **TokenDelay**. At ADC level it depends also from the cable delays. This check is valid if the trigger is issued while the token chain is idle (not processing a previous event).

  2. The time between two consecutive events is different from **TokenDelay**.

  This check is valid if the trigger is received while the chain is processing a previous event, i.e. when the actual event has to wait in the output queue.

  3. The total time of the DataValid for a single event (**DataLength**) is wrong.

  This time depends on the number of HELIX in the token chain, so it can be chain-dependent (barrel vs. wheels, disabled chips)

- <u>Pipeline out of phase</u>

  Within each token chain, the number of the column in the pipeline has to be the same for all chips. The ADC will check the numbers (8-bit trailer of each chip) and report error if there is a mismatch.

At least one parameter (DataLength) has to be reprogrammable in the ADC, to cope with situations where chips are disabled in a chain. The TokenDelay has to be downloaded into the chips, but there is no reason to change it once defined. Also the DataLatency will be fixed (but it will change with TokenDelay).

Handling of the ADC error condition

The idea is to be able to reinitialise the HELIX chains and restart. In some occasion it will be necessary to disable a chip (if the culprit is known or traceable) or a complete ADC unit. In any case these operation have to be performed with the system idle (GFLT busy). To achieve this the ADC, C+C and CPU have to be involved.

Just re-init

It is conceivable to find error conditions which can be corrected just by a simple reset of the logic (i.e. a chip has missed a clock or a trigger). This is the easier way to try a recovery, and probably is the first thing to try in case of error:

The ADC module with an error condition will:

- Write an appropriate code (to be defined) in the output data.
- Assert the **error request** line, which via the C+C interface will reach the C+C master. This is a latched line.
- NB: all the event still in the FIFO of the bad ADC unit have also to be marked with an error status.

On error request the C+C will:

- Assert the GFLT busy, at the next available FLT.

The crate CPU will:

- Identify the error condition. How? From the error code in one of the ADC, or directlly from an interrupt generated by the C+C interface.
- Verify that the system is actually busy, so that no extra event can be expected. This is needed if one has to reinitialise the pipelines. The busy status has to be echoed by the C+C in a VME register of the C+C.
- Go on reading all the events from all the ADC. This can be verified from a “FifoEmpty” signal from each of the ADC.
- Reset the error-request latch.

On the trailing edge of the error request the C+C will:

- Issue a NotReset (pipeline reset) to all the HELIX chains
- Resume running (release busy to GFLT)

Some questions on this machinery:

- Is the error request latched at ADC or C+C interface level? Probably at ADC, it can be used then to mark as bad the events in the FIFO.
- Is the FIFO-empty a VME register in the ADC with a bit per ADC unit (just test if 0)? Or a wired-OR of all ADC at C+C interface level (to speedup checks done by the CPU)?

If re-init is not enough

If a chip is damaged, it will keep making the same error and the previously described recovery operation will just be called every event (or second event). So an error rate monitor is needed, and possibly some part has to be disabled. The present BMUON strategy (quite rough) is:

- Keep a global error counter and a local error counter for each item that can generate an error.
- After N global errors, or after some time interval, check if there is a local error counter that has too many errors (close to N).
- If so, disable that item and in any case reset all the counters (global and local).

What to disable

The problem in our case is what to do if the error persists

1. Disable a single chip
2. Disable an ADC unit
3. Abort the run

1. Disable a single chip:
   - If it is known (for instance, goes always out of sync):
     - Tell the C+C to reprogram the token ring to actually exclude the chip. How? Use a network connection and issue a specific reconfiguration command to the CPU of the HELIX-interface crate. Or write the new configuration into the (shared) database and tell C+C just to reconfigure everything. Then reprogram the ADC with the new configuration for the chain.
   - Search for a dead chip and disable it. Here, I really don't know where to start.
2. Disable a complete ADC unit. The CPU reprogramming the ADC can do this. To be defined how by Katsuo.
3. If the problem is not solvable, or too many things have been already disabled, the CPU has to abort the run. How? Using the Fatal Error signal (on the C+C interface) or via run control?

In any case, a further problem is how to keep track of all this. One can imagine that the initial configuration is sent with the BOR event. I see few possibilities:

1. Send always some header info together with the data (for every chip, or token chain?). This is in Katsuo's proposal (control data), but with 20 KB/event it is too big.
2. Send the information only for chains with some error condition
3. Or only in the environmental events (BTW, does the ADC see the ENV event as test triggers, with some special readout type?). In this case, what happens in the (short) time between the reconfiguration and the next ENV event?

And the main issue is what to do at SLT level to cope with missing chips, chains etc

Errors not from the ADC

We can have errors not reported by the ADC, but that anyway deserves the same kind of treatment?

1. Error at chip level. We can have ringing strips or chips that provide nonsense data. If data is going to be used at SLT, it is probably useful to be able to mask channels during the run.
2. An ADC unit may go bad, without reporting any error condition. Then, one can try to disable it.
3. Clearly a complete ADC can go bad, but I doubt we can run without an ADC.

In any case, it is needed (or at least cleaner) to keep the busy while one reconfigures. So, the C+C interface needs a register where the CPU can set the error request line.