Version 1.0

A description of a smallish part of the BepiColombo (BC) MIXS/SIXS Onboard instrument SW.

In short, the following explains telecommand (TC) reception, and TC acknowledgement (service 1 of the PUS standard – see below for details) the way it is done within the BC MIXS/SIXS instrument SW. A suitable subset of what is presented (in prose) below, may be chosen. This – possibly too – "large" description is given since it's difficult to isolate anything smaller without putting things into a context. Naturally parts of the functionality presented may simply be abstracted away, as preferred.

TC reception

The BC MIXS/SIXS instrument SW receives telecommands (TCs) via a dedicated hardware link. These TCs are (by means of HW) placed in a dedicated area of memory that is treated as a circular buffer. The HW administers (meaning, only the HW has write access to the pointer, although both HW and SW can read it) a write pointer, which points to the first "writeable" memory word. The SW administers a read pointer, which points to the first unread memory word. The write pointer may not "overwrite" the read-pointer, i.e. if the SW cannot process incoming TCs fast enough, TCs are simply dropped.

The pointer administration has to – naturally – take into account proper wrap-around handling.

The SW polls the buffer at a pre-defined rate (the activity is planned to be non-interrupt driven, since that gives a better control of the schedulability of the overall system), and (based on difference of read and write pointers) determines whether any TCs are present. If the read and write pointer values differ then the responsibility of the SW is to read out info on the length of the packet (see below for details), and determine whether the packet is completely received by the HW (based on write pointer value), and if so, read out ONE TC packet from the buffer (potentially there might be more than one TC in the buffer, but the SW's responsibility is to manage a pre-defined rate of TCs, reflected in the polling rate), which is copied into the *TC/TM pool* (see below), and the read pointer is updated accordingly.

After this the TC is passed on to a separate task for further delivery, to the actual recipient.

TM delivery

Similarly as for TC reception, there is also a TM delivery service (as the TC reception, also this function is operated at a pre-defined rate). Here the HW is made aware of a need for TM delivery via a number of "HW registers" (technically these can be seen as specific memory addresses from the p.o.v. of the SW). There is an address register, a length register and a status register. The address register is used to indicate to the HW the address of the first word of the TM packet. The length register contains the TM packets length in bytes (or octets, which is the term used in the space business). The status register is used to signal the HW that a new packet is ready for transmittal (and that the address and length registers have been accordingly updated). In practice this means setting a dedicated bit in the Status register to "1". Once the HW has finished the transmission of the packet it sets the corresponding bit to "0". The status register can also communicate other information between the HW and the SW. For instance, the HW might indicate transmission errors by setting some other bit in the status register, whereupon the SW may consider whether to re-transmit, or simply drop the packet. Once the SW is ready with the packet, it has to inform the TC/TM pool that the packet "container" in which the packet resides, can be released (see below).

Format of TC (and TM) packets

The TCs follow a strict syntax (based on the European Space Agency's Packet Utilisation Standard [PUS]). TCs are divided into two different major parts: "Packet Header" (which contains a rather large set of data, including the length of the packet, the APID – see below, and a large number of other information that is irrelevant in the scope of this presentation) and "Packet Data Field". The Packet Data Field itself is divided into a (data field) header (determining among other the type and subtype of the packet) followed by the actual data followed by an error control field.

In response to TCs, the receiver may issue telemetry (TM) reports. Similarly to TC packets, also TM packets follow a strict syntax. The top-level structure of TM reports is roughly the same as the structure of TCs. TM reports may also be issued by an application spontaneously (e.g. in case some predefined event occurs, which may require intervention outside of the application issuing the report).

PUS divides the services provided into types and subtypes. E.g. memory management (service-type 6 in PUS) forms one service. Memory management includes such subservices (sybtypes) as patching (the data in the "patch" TC is to be written into an address that is also explicitly identified in the data of the TC), or dumping (the start address of the memory area to be dumped is provided as part of the data of the "dump" TC as is the length of the memory area to be dumped). A type/subtype pair is always unique, i.e. if a TC(5,19) is defined, no TM(5,19) report may be specified.

TC and TM packets both have a minimum and a maximum size. However, the maximum size of a TM (roughly 4000 bytes) packet is roughly 20 times that of a corresponding TC packet. Minimum sizes are almost equal (roughly 10 bytes).

TC checking

On reception of a TC, the TC is checked for errors w.r.t. the specified standard. The following – minimum set – of errors at least need to be checked:

- APID check (the Application Process ID identifies the recipient – among potentially many; MIXS/SIXS will define at least 5 independently "scheduled" TC-receiving tasks or APIDs – of the TC)
- Length check (the length of a TC is indicated in the Packet Header)
- Check of checksum (error control at the end of the packet)
- Validity of packet type (in data field header)
- Validity of packet subtype (in data field header)
- Validity of application data (type + subtype dependant)

If any of the above checks fail, a TM report indicating the failure should be issued, so that ground or any go-betweens may know of the reason for failure. It's worth noting, that – depending on the architectural solution chosen – some of the above checks may not actually be performed (e.g. subtype, and more specifically, application data validity) by the *primary* TC receiving process, but may rather be (read: will be – it is almost certain that the actual approval of a TC for execution will be done by the actual recipient, rather than the global central TC checker) checked by the TC-recipient of the APID to whom the TC is directed (Space System Finland has – as of yet – not decided exactly how to implement this particular aspect). In any case, this part (the final approval) and actual execution of TCs can simply be seen as a non-deterministic black-box, at this stage.

Service 1

The PUS service 1, is the service used to acknowledge TC reception. Regardless of the service which a given TC received by the SW is belonging to, it is acknowledged by means of one or more (two in the BC case – but generally there might be more) service 1 TM reports. This service is rather special, in the sense that it only defines TM packets. It's not controlled by TCs at all. The reports use (i.e as part of their application data) data copied from the TC packet header to allow ground to identify the originating TC. The BC relevant TM reports are the following:

- TM(1,1) – TC Acceptance – Success. Issued if the TC is verified to be "syntactically" correct (no failures in the above sketched checks). The TC is accepted for execution of the addressed APID.

- TM(1,2) – TC Acceptance – Failure. Issued if the TC is verified to be “syntactically” incorrect (some failure[s] in the above sketched checks – first identified failure reported). The TC is rejected. The TM(1,2) report contains a Failure ID (FID) identifying the reason of failure.
- TM(1,7) – TC Execution – Success. Issued if the TC was “semantically” correct, and execution finished successfully.
- TM(1,8) – TC Execution – Failure. Issued if the TC was “semantically” incorrect (e.g the command was “syntactically” correct, but for some reason execution could nevertheless not be started, due to the context), or execution finished unsuccessfully (e.g. the execution was started but for some reason couldn’t finish successfully). The TM(1,8) report contains a FID identifying the reason of failure.

Note that the exact boundary between “syntactic” and “semantic” failures is rather vague.

It is an overall design goal, that the service 1 services will be encapsulated in their own module, and provided to user’s via a well-defined “API”. The API driven module, should create the TM report via the TC/TM pool (see below), and route it properly for further transmission. The API should not require the users to have to access the TM packet at all, rather, just have them provide the data necessary to fill out the reports through parameters in the used function calls to create the report. The releasing of the occupied TM “container” (see below) is the responsibility of the TM delivery service (see above).

The TC/TM pool

There will be a TC/TM pool available for the BC MIXS/SIXS SW, that offers a centralized service for the intermediate storage of TC/TM packets under checking/execution/generation. This will be a self-contained SW module offering a well-defined API to the tasks of the SW. The main purpose of this service will be to avoid unnecessary duplication/copying of data (ideally any TC/TM packet “read into/created by,” the SW should never have to move, under the unavoidable restriction, that memory may NOT be allocated dynamically).

There are reasons why one could actually want to make this pool rather sophisticated (e.g. a “poor mans” version of a “paged/segmented” memory, with sufficiently small “pages/segments”, in order to save memory), but current (this is still not fully known) memory budgets seems to indicate that this is not necessary. Thus, the pool needs to maintain sufficiently many maximal size “containers” for TC/TM packets under execution/creation in order to sustain requirements (what this means in practise is open for the time being). Given this approach, once a TC is read, or a TM packet is created into the TC/TM pool, it does never actually have to move, while being processed by the SW, and e.g. TC/TM queues can be implemented as queues of pointers.

Necessary API-provided functions are functions to read/write from/to “containers” (via a pointer). Also operations to set the read/write pointer to specific positions in given “containers” and to release a given “container” for re-use.