

**Flight Software for xDR and JINx nodes**  
**in AMS-02**

A.Kounine and V.Koutsenko

24 February 2011

Comments to Andrei.Kounine@cern.ch

# Table of Content

1	Introduction .....	5
2	DAQ system design considerations .....	6
3	Block diagrams of the nodes .....	10
4	Power-up procedure .....	12
5	FLASH file system.....	13
6	AMSWire operations .....	15
6.1	Command Header .....	16
6.2	0-length AMSWire packets.....	17
6.3	Reply formats.....	18
6.4	Group Command assembly.....	19
7	Processing of Commands.....	21
7.1	Boot Command .....	21
7.2	Read Node Status .....	21
7.3	Ping Command.....	24
7.4	DSP Power Down Command.....	24
7.5	Program Test.....	24
7.6	FLASH File Read .....	24
7.7	FLASH File Write.....	24
7.8	FLASH File Test.....	25
7.9	Load FLASH File .....	25
7.10	FLASH summary Read.....	25
7.11	FLASH File/Sector Erase .....	25
7.12	File Attribute Set.....	25
7.13	Perform IO Command.....	26
7.14	Program Memory Read.....	26
7.15	Program Memory Write .....	26
7.16	Data Memory Read.....	26
7.17	Data Memory Write .....	26
7.18	Read Event.....	26
7.19	Read Last Event Number .....	27
7.20	Reset Event FIFO.....	27
7.21	Read Housekeeping Info.....	27
7.22	Node Configuration Read Command.....	28
7.23	Node Configuration Write Command.....	28
7.24	Processing Mode Read Command .....	28
7.25	Processing Mode Write Command .....	28

7.26	Calibration Status Command .....	28
7.27	Calibration Control Command.....	28
7.28	Sub-Detector Procedure Status .....	29
7.29	Sub-Detector Procedure Control.....	29
7.30	Slave Test Status .....	29
7.31	Slave Test Control.....	29
7.32	Slave Mask Read.....	29
7.33	Slave Mask Write.....	30
7.34	BUSY Mask Read.....	30
7.35	BUSY Mask Write .....	30
7.36	DELAY value Read .....	30
7.37	DELAY value Write .....	30
7.38	BUSY Status Read .....	30
7.39	BUSY Errors Read.....	31
7.40	SSF Status Command .....	31
7.41	SSF Set Command .....	31
7.42	QLIST Entry Delete.....	31
7.43	QLIST Read Command .....	31
7.44	QLIST Write .....	32
7.45	LeCroy Bus Read.....	32
7.46	LeCroy Bus Write.....	33
8	Node Initialization and Configuration .....	33
8.1	Configuration File.....	34
8.2	AMSWire Configuration Command.....	34
8.3	Configuration of JINF and JINJ nodes .....	34
9	Calibration Procedure.....	35
9.1	Internal Calibration .....	37
9.2	External calibration.....	38
10	Sub-detector Specific procedures .....	38
11	Event Building Procedure in CDP .....	38
11.1	Event Processing in CDP .....	40
12	Event Building Procedure in CDDC .....	40
12.1	Event building format in JINJ.....	41
12.2	Event building format in JINF .....	42
13	Slow Control Procedure.....	42
14	Data an Program Memory usage .....	44
15	Buffer Memory usage .....	45
16	Data Protection .....	46

16.1	FCS Calculation in Gate Array .....	46
16.2	C-code calculation of FCS .....	47
17	External AMSWire interfaces .....	48
18	Conversion Program for FLASH File System.....	48
19	Availability of the code and documentation.....	48
20	Appendix.....	49
20.1	ADSP-2187L Register Set .....	49
21	References.....	52

## 1 Introduction

AMS-2 Data Acquisition System collects data from over 200K analog channels of the different AMS-2 sub-detectors: TRD (U), TOF and Anti-Coincidence Counters (S), Tracker (T), RICH (R), Electromagnetic Calorimeter (E) and Level-1 Trigger module (LV1). It consists of nearly 300 computational nodes based on ADSP-2187L Digital Signal Processors [1] and a Main DAQ Computer based on PPC750 processor.

DAQ architecture has a tree-like structure: 264 xDR nodes (DR -Data Reduction, x - specifies a sub-detector: T, U, R or E) collect data from analog electronics Front-Ends; 28 JINF nodes collect data from xDR nodes; 8 SDR nodes collect data from TOF/ACC and produce trigger signals and 2 JLV1 nodes collect analog and digital information to produce LV1 trigger; and 4 JINJ collect data from JINF+SDR2+JLV1 nodes. The nodes are interconnected with point-to-point LVDS serial links. AMSWire protocol [2] is used for communication. The data throughput per link is 100 Mbits/s.

The software for ADSP-2187L is designed as a sub-detector independent framework and a set of detector dependent data processing routines. The present note describes the software corresponding to version 0xAB06 or higher. Main ingredients of the framework software are the following:

- ROM monitor program to allow for several boot options;
- FLASH update utility to store and retrieve information between power cycles;
- AMSWire protocol for communication between nodes;
- data protection based on CRC algorithm;
- slow control procedure to gather and keep up-to-date information on the node state;
- a set of test routines for node functionality testing;
- framework routines for HW initialization/configuration;
- a set of routines to perform calibration;
- event building routines for physics event assembly.

Detector specific routines are relevant for xDR and JINF nodes, where the raw physics events are processed and detector specific initialization is performed. The interfaces to the framework software are common for all the sub-detectors.

The AMS experiment will operate at trigger rates up to 2 kHz, an average event size being about 2KBytes. The maximal Front-End readout time is 90 ns, which corresponds to a dead time of 16% at 2 kHz. The goal for DAQ design is not to increase the dead time due to data processing. Therefore the DAQ parameters are optimized for that and performance issues at high trigger rates are also addressed in the present note.

## 2 DAQ system design considerations

A block diagram of the DAQ tree is shown in Figure 1. Hierarchy in the system is defined by the master-and-slave communication protocol which establishes the following relations between the nodes: each xDR has no slaves, and has two JINF masters; each JINF has up to 24 xDR slaves and 2 JINJ masters; and each JINJ has 20 slaves (14 JINF, 4 SDR and 2 JLVI) and 4 JMDC masters. A slave sends no data to its masters without a request and responds to a master request as quickly as it can and in any event within about 1.2 second timeout period.

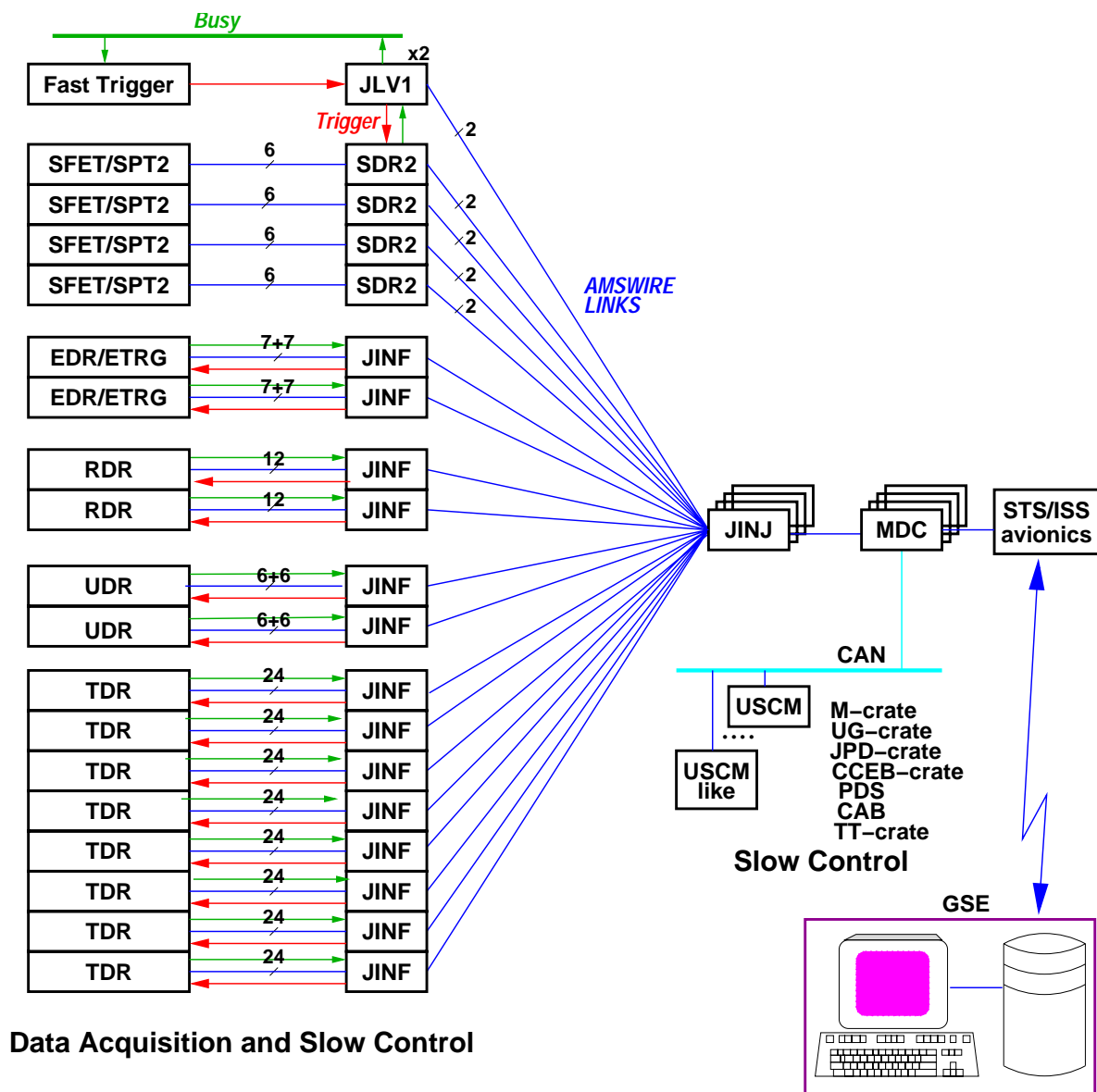
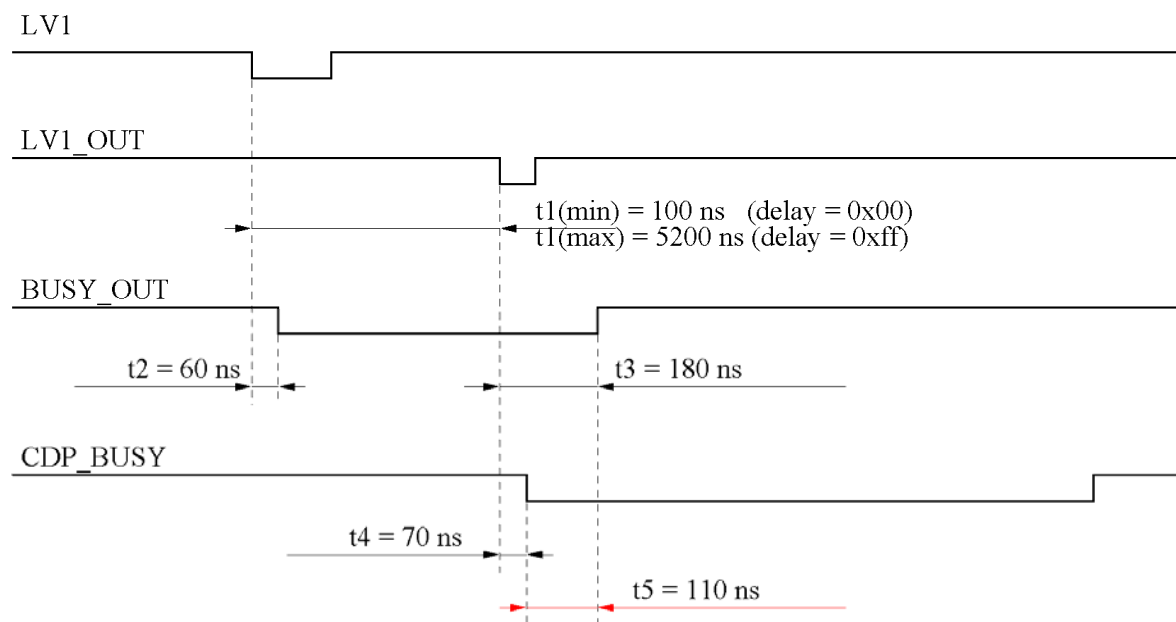


Figure 1 DAQ Block Diagram.

The baseline operation mode assumes that there is no redundancy in xDR nodes (however in the current implementation UDR, EDR and SDR2/SPT2/SFET nodes are double redundant), there is double redundancy in JINF nodes and there is quadruple redundancy in JINJ nodes.

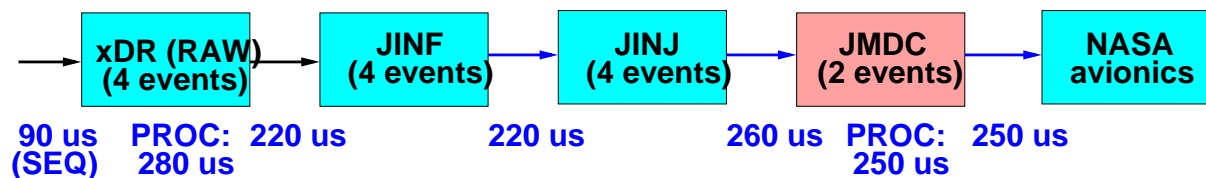
Physics event building principles are the following: each node collects, processed and stores physics event fragments in an internal buffer independent on other nodes. Event fragments built by a node are retrieved by the node master using a corresponding AMSWire command. This is the only AMSWire command issued by JINF and JINJ nodes on their own.

LV1 trigger signal is generated by the JLV1 module based on the analog information from ToF, ACC and EMC systems. This signal is distributed to all xDR nodes via dedicated circuitry on JINF boards. Upon arrival LV1 trigger signal to an xDR board a dedicated GA logic starts digitizing analog information from the FE electronics and stores it in the CDP Buffer Memory. During the digitization it also asserts BUSY signal which is delivered to JLV1 node. Timing diagram showing relation between signals generated by deferent nodes is presented in Figure 2. Logical OR of all BUSY signals, when asserted, vetoes generation of LV1 triggers. Therefore the digitization time (up to 90µs for TDR nodes) defines the system dead time at high trigger rates. At 2 kHz it amounts to 16%.



**Figure 2** Time diagram of LV1 and BUSY signals set by different nodes. LV1 signal is generated by JLV1. In response to this signal JINF generates LV1\_OUT according to the programmable delay, and BUSY\_OUT signals. CDP\_BUSY is set by an xDR node. During the time interval ( $t5$ ) all CDP BUSY signals are checked and corresponding bits are set for those missing.

Data flow in the system is schematically shown in Figure 3. On LV1 trigger signal an event fragment corresponding to a sub-detector part is moved by the sequencer into the raw event buffer in xDR. It is then processed by DSP and stored into the built event buffer of xDR. Next it is retrieved by JINF using an AMSWire command, assembled with other xDR event fragments by DSP and stored in the built event buffer of JINF. This newly built event fragment is retrieved by JINJ with an AMSWire command, assembled with other JINF event fragments by DSP and stored in the built event buffer of JINJ. Finally events from JINJ are retrieved by JMDC for final assembly and LV3 analysis. After LV3 filtering they are sent to NASA avionics and downlinked. Typical measured timings are shown in Figure 3. The event building procedures in JINF and JINJ are identical.

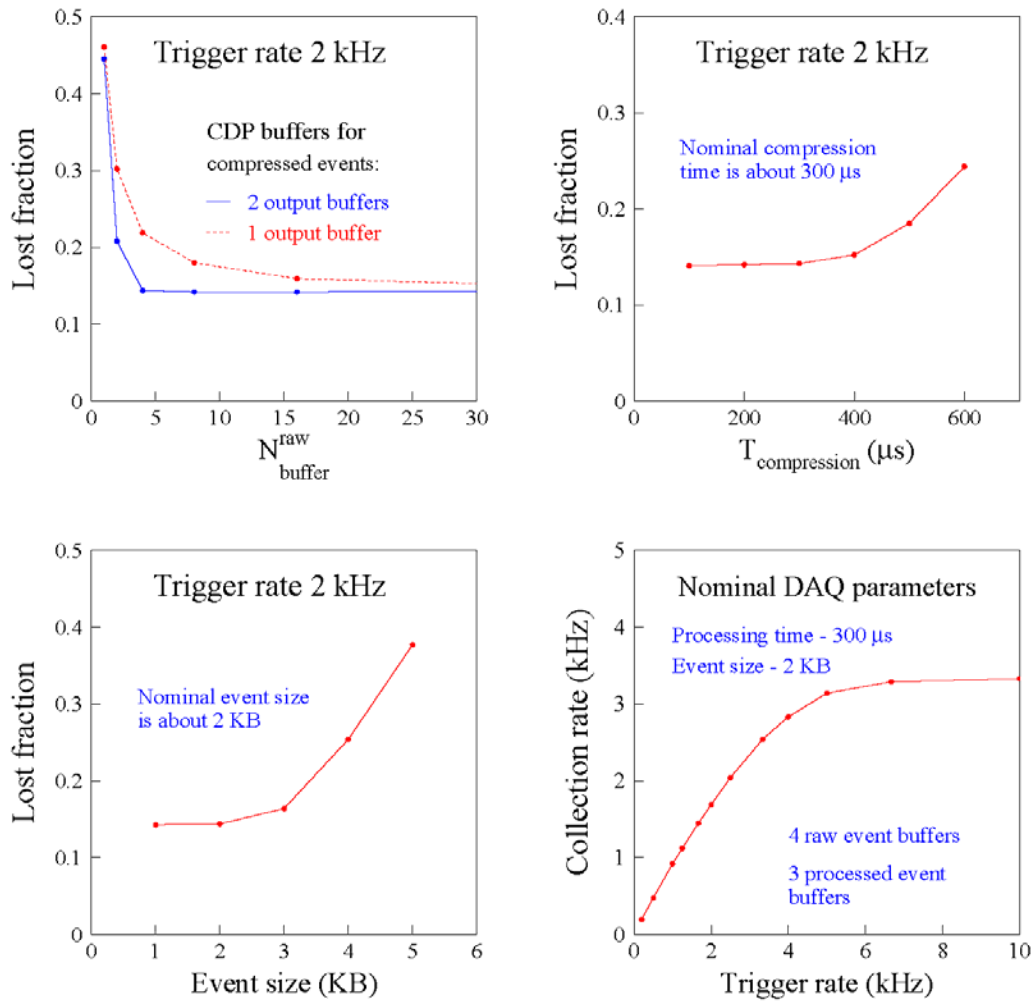


**Figure 3** Data flow and typical event building timing in the AMS-2 DAQ.

In order to minimize extra dead time due to data processing in the DAQ system event buffering is used at every level of DAQ hierarchy. To evaluate the optimal parameters a Monte-Carlo study of the system performance was done using a benchmark Fast Trigger rate of 2 kHz. Figures 4a–4c show the system dead time as a function of most important parameters: buffer depth for raw and built event buffers; event processing time in xDR nodes, and average event length. From these studies the following values are adopted for the system design:

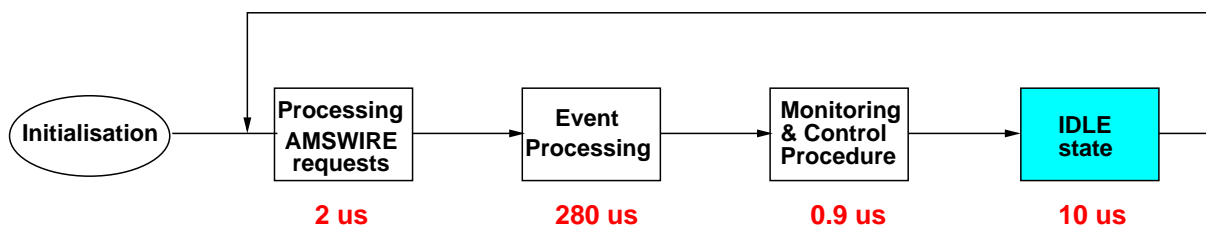
- Average event size – < 2.5KByte;
- xDR raw event buffer size – 4events;
- xDR processed event buffer size – 4events (8 for SDR2 and JLV1 nodes);
- xDR event compression time – < 280 $\mu$ s;
- JINF event buffer size – 4events.
- JIM-AMSWIRE buffer size –  $\geq$  2events.

These settings are used to evaluate the physics event collection rate as a function of Fast Trigger rate. Results are presented in Figure 4d.



**Figure 4** DAQ performance as a function of critical parameters.

Information on the node state (Monitoring and Control Procedure) is collected at specified time intervals (on a scale of minutes) and is kept in memory. This information may be retrieved by the node master at any time. Most general block diagram of the program flow is presented in Figure 5 along with the corresponding average timings.



**Figure 5** Program flow concept for xDR/JINF/JINJ nodes and typical timings at highest trigger rates. In the infinite loop all the boxes are optional, except servicing AMSWire requests.

### 3 Block diagrams of the nodes

Both xDR and JINx nodes are based on ADSP-2187L digital signal processors. The basic concepts of the architecture are outlined in Figures 6 and 7 for xDR and JINx nodes, respectively.

Operations of xDR and JINx nodes are centered around SRAM buffer memory which is accessed independently by DSP, AMSWire RX/TX and by the FE sequencer which collects information from the front-end electronics. A typical sequence of the event processing operations in an xDR node is the following. First, on LV1 trigger the sequencer starts moving digitized data into the buffer memory. Next, when DSP sees that the data are available, it reads and processes the data and stores the result (i.e. a processed event fragment) in the output event buffer in BM. Finally, on a corresponding AMSWire request (a data block written into the buffer memory by the AMSWire Receiver and then read out and analyzed by DSP) the processed event fragment is transmitted from the buffer memory by the AMSWire Transmitter.

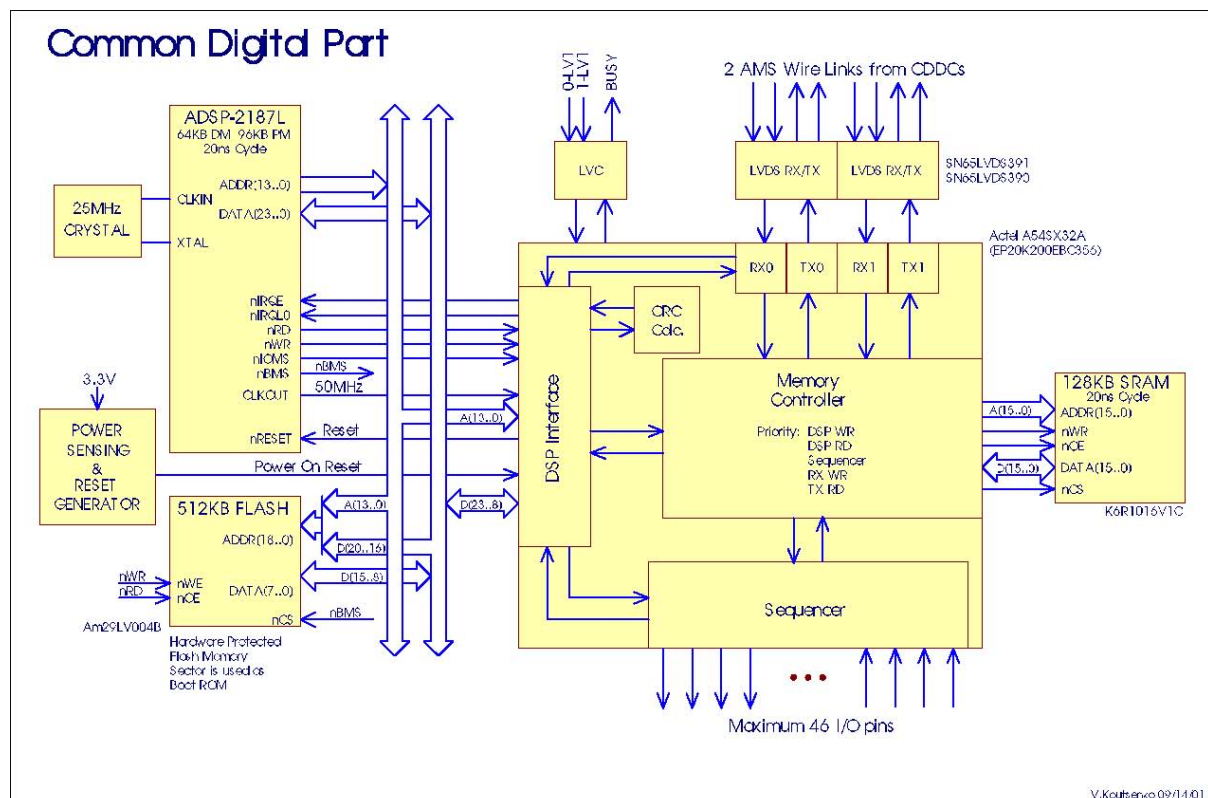
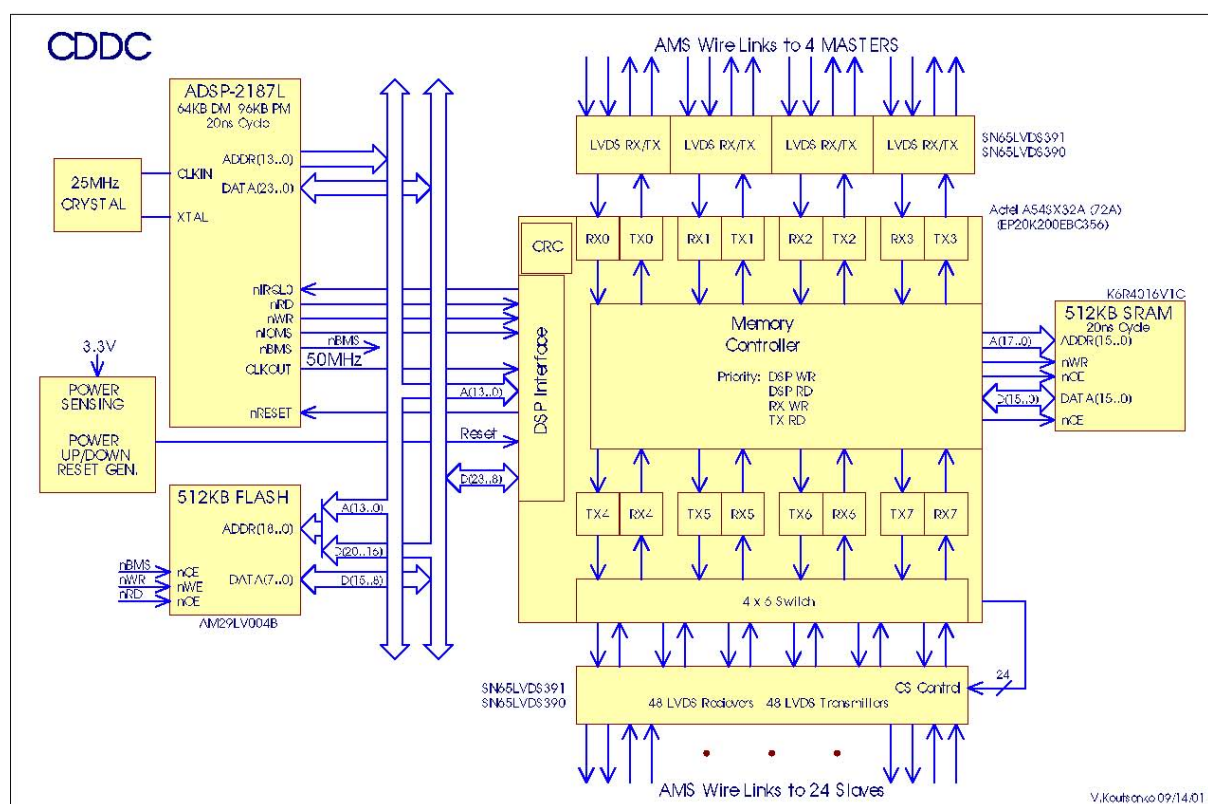


Figure 6 Block Diagram of a CDP node.

LV1 and BUSY chain as well as AMSWire Receivers and Transmitters function independent on the DSP operation. This imposes restrictions on the frequency of DSP accesses to the buffer memory. The rule of thumb is that DSP should not attempt to take more than 50% of the SRAM bandwidth, i.e. only one memory access by DSP in 60 ns.

8 LeCroy buses are connected to each of JINF nodes. They are driven by the DSP serial port running at 500 kHz.

Only three external interfaces are foreseen for programming and debugging purposes: JTAG for the FPGA testing, EZ-ICE for the DSP programming and debugging, and AMSWire for communication with the node.



**Figure 7** Block Diagram of CDDC node.

## 4 Power-up procedure

Power-up procedure is several step procedure common for all the ADSP-2187L based nodes. At the end of this procedure DSP executes a program capable to communicate with the master node using AMSWire links.

First, a bootstrap code – first 96 bytes at the very beginning of the FLASH memory – is loaded to the ADSP-2187 Program Memory starting at address 0x0000 and execution starts at address 0x0000. The bootstrap code loads boot loader to Program Memory.

Boot loader loops over 3 identical copies of ROM Monitor– a simple and robust program with minimal functionality. Each of the three is protected by its own Frame Check Sequence (FCS) calculated using 16-bit CRC algorithm. The FCS is checked by the boot loader and the first one that passes the test is loaded to PM address 0x0000 using BDMA transfer with context reset at the end of BDMA transfer. In case of the triple CRC failure the majority logic is used to construct a new ROM Monitor out of three available versions. Thus at the end of this procedure the program control is passed to ROM Monitor one way or another.

After initialization of the ADSP-2187 registers, ROM Monitor looks for a presence of boot command. In the event that boot command is found, the boot parameters are read-out and ROM Monitor proceeds accordingly – loads specified programs from FLASH memory. In the event the boot command is not found ROM Monitor loads a default program, if any. If there is no default program ROM Monitor continues running and performs only servicing AMSWire requests. In case of detected RX errors associated with the boot command ROM Monitor does not attempt loading any other program.

**Table 1. FLASH memory map.**

Address range (bytes)	Sector	Usage	Comments
0x00000–0x00060	0	bootstrap code	HW protected
0x00061–0x001FF	0	boot loader	HW protected
0x00200–0x02BFF	0	ROM Monitor, v1	HW protected
0x02C00–0x055FF	0–1	ROM Monitor, v2	HW protected
0x05600–0x07FFF	1–2	ROM Monitor, v3	HW protected
0x08000–0x0FFFF	3	a file	not protected
0x10000–0x1FFFF	4	a file	not protected
0x20000–0x2FFFF	5	a file	not protected
0x30000–0x3FFFF	6	a file	not protected
0x40000–0x4FFFF	7	a file	not protected
0x50000–0x4FFFF	8	a file	not protected
0x60000–0x6FFFF	9	a file	not protected
0x70000–0x7FFFF	10	a file	not protected

The allocation of FLASH memory space is detailed in Table 1. The bootstrap code, boot loader and ROM monitor are located in the first three sectors of the FLASH memory. This arrangement limits the maximal size of the ROM Monitor program to 0xE00 instructions. ROM monitor consist of even number of instructions (DSP instructions are 24bit-wide), including FCS. This is achieved by adding one or two NOP (i.e. 0x000000) instructions to the actual code and filling the last two bytes with the FCS corresponding to the rest of the program. The first three sectors of FLASH memory are HW protected such that their content can not be modified by a program. Other FLASH sectors are not protected. They host a file system to keep programs and data sets.

The first time FLASH programming is done using EZ-ICE. This option is especially useful for the program development phase. Using this approach, ROM Monitor (or any other program to be written to FLASH) is loaded to PM AAC and bootstrap code, boot loader and a dedicated FLASH Programming utility are loaded to PM OV0 using EZ-ICE emulator. Write parameters (size of all the programs) are hard-wired to the FLASH Programming utility. Execution is passed to the FLASH Programming utility by setting the DSP Program Counter by hands (one may also use a TCL script). It calculates the FCS value for the ROM Monitor code and appends it as the last “instruction” and writes it to the FLASH memory three times at addresses specified in Table 1. Another option for the first time programming is programming FLASH before soldering using a dedicate programmer.

During normal operation a new program is loaded into FLASH using a correspondent AMSWire command. Upon receiving such a command an internal FLASH Update utility decodes the command, extracts the file and programs the FLASH memory accordingly.

## **5 FLASH file system**

Maximum of 8 files (corresponding to 8 unprotected FLASH sectors) is allowed. Each file is identified by one-byte file name and characterized by the file type (program or data) and file attribute (default -non-default file). File type defines whether the program that loads the file will be overwritten by the file content (i.e. BDMA with context reset) or not. File attribute defines whether the file will be loaded at initialization or not. Memory type in the segment header defines three possible destinations for the data in a segment body: PM, DM or IO. Segment length corresponds to the number of words in the segment body. The word width depends on a memory type – it is 3 bytes for PM segments and 2 bytes for DM and IO segments. The length of a PM segment is required to be even. An IO segment consists of pairs Register ID and Register Value. Bits 11–0 of Register ID specify the register ID and bit 12 specifies READ (0) or WRITE (1) operation. The segment body is protected by 16-bit CRC algorithm irrespective of the segment type. Frame Check Sequence is included in the segment length calculation.

Flash directory information is retrieved from FLASH memory at initialization and stored in the DSP Data Memory. It is updated after every operation with FLASH -writing a file to FLASH, testing a file, deleting a file or erasing a sector or modifying file attributes. This allows retrieving information on FLASH directory at any time (even when sector erase operation is ongoing). The file structure is presented in Figure 8. A file consists of the file header and up to 14 segments. A segment header contains the information needed to set up the corresponding BDMA transfer. It includes the segment length, type of memory (PM – 3-byte wide; DM/IO – 2-byte wide) and load address.

## FILE HEADER

<b>FT</b>	<b>FA</b>	<b>File name</b>	<b>N segments</b>
14	12		4

**File type FT:** 00 – Program  
 01 – Data set  
 1X – reserved

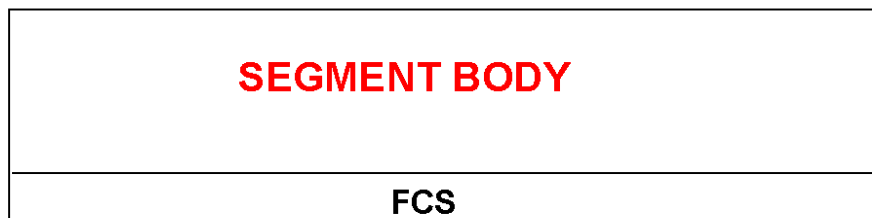
**File attr. FA:** 11 – A file  
 10 – Default file  
 0X – A file

## SEGMENT HEADER

<b>MT</b>	<b>Segment length (words)</b>
<b>PP</b>	<b>Load address</b>
14	0

**Memory type MT:** 00 – PM  
 01 – DM  
 1X – IO

**PM/DM page PP:** 00 – Page 0  
 10 – Page 4  
 11 – Page 5  
 01 – invalid



**Figure 8** Structure of a FLASH file.

File (or sector) is deleted with “Erase FLASH File/Sector” command. The measured timing for this operation ranges between 0.8s and 1.4s, but may take as long as 15s according to the AM29LV004B data sheet. Therefore the AMSWire command “Erase FLASH File/Sector” just initiates the erasure procedure. When this procedure is in progress, no other FLASH erase/program/read array commands may be accepted. Thus any request for writing to FLASH or loading from FLASH will be aborted, if received when FLASH erase operation is in progress. The current status of the FLASH file system (including the status of FLASH erasure operation, if any) can be controlled with “Read FLASH Summary” command.

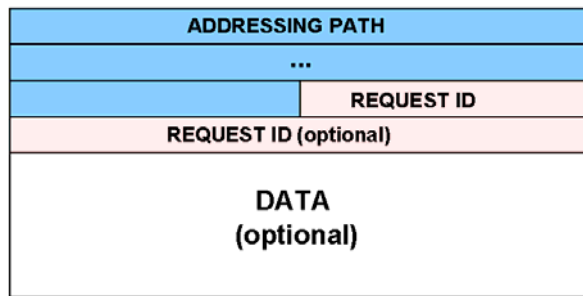
File attribute (default/non-default) may be changed at most two times during file lifetime: one time from non-default to default and once from default to non-default. In latter case the corresponding file may not be loaded anymore by the initialization procedure. It still may be loaded by the “Load FLASH File” command.

## **6 AMSWire operations**

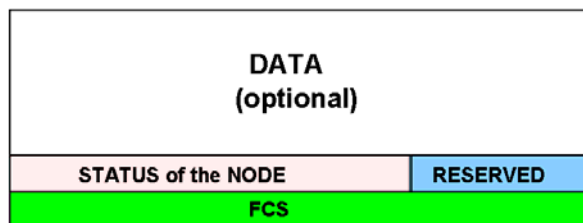
AMSWire protocol discussed in this section is a high-level communication protocol between xDR and JINx nodes. It defines master-slave relations between the nodes. Each node has several masters and may have some slaves. A slave node receives and services master requests as quickly as it can. Typical AMSWire link throughput is measured to be 7.6 MByte/sec for long (> 1kByte) blocks. Master sends requests to slaves at its own convenience.

xDR nodes have no slaves. JINF nodes have several xDR nodes as slaves and two JINJ nodes as masters. JINJ nodes have 24 JINF/SDR2/JLV1 slaves and 4 JMDC masters (see Figure 1). Each of xDR/JINF/JINJ nodes has knowledge of its immediate masters and slaves only. In other words a node does not have a knowledge of which part of DAQ hierarchy it belongs to. Only JMDC node (not discussed in this note) has the entire map of AMSWire connections. General format of AMS data blocks is discussed elsewhere [3]. Here we discuss specifics of AMSWire implementation of the general scheme. Each AMSWire command consist of a header which contains addressing path and command ID and, optionally, data part. Replies have only the data part. The last two words of any reply are Reply Status and CRC check sum. In the Reply Status word only bits 10-5 are filled by the replying node. The other fields are filled by the node master during the group command assembly or event building (see Section 7.2 for details). Format of AMSWire requests and replies is presented in Figure 9.

## LONG REQUEST FORMAT:

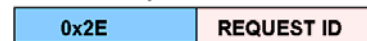


## LONG REPLY FORMAT:



## ADDRESSING SCHEME

### Individual Request



### Request to a slave



### Group Request A



### Group Request B

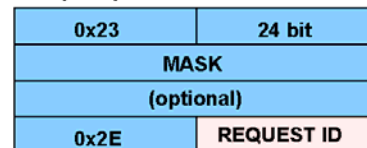


Figure 9 Structure of AMSWire commands and replies

### 6.1 Command Header

Each AMSWire command (AMSWire Block) is a request to a node to perform a specific action which may or may not require communicating with the node slaves. A command may request a node to perform read/write operation on a media belonging to the node; or to pass that command to one of its slaves and direct the slave reply to the master which sent the original command; or to send the command to several slaves, collect and assemble replies in one data block and send this data block as a reply to the original command. Specific of these three different request types is reflected in the addressing scheme of the AMSWire requests (presented in Figure 9).

Information contained in the first byte of the command header is used by the recipient node to choose one out of three procedures for command processing:

1. Command destination is the node itself: first byte is 0x2E. The node analyses Command ID and executes the command;
2. Command destination is a slave of the node which processes the command: first byte is in the range from 0x00 to 0x17, corresponding to the slave ID. Second byte defines the AMSWire port to use (4, 5 or 0x3F. The latter value means any port can be used). The node strips off the first word from the header and sends the remainder to the corresponding slave;

3. Group command A: first byte is 0x40. The list of the slaves belonging to the group is defined by the Mask ID contained in the second byte. This mask should be already defined. The node strips off the first word from the header and sends the remainder to the specified slaves. Then it collects replies, assembles them and, upon assembly completion, sends the result to the master which made a request;
4. Group command B: first byte is 0x23. The slave mask is defined by the following three bytes.
5. Group command C: first byte is 0x2A. The request is sent to all the slaves.
6. Any other value of the first byte represents an error.

The structure of the request ID (see Figure 9) is defined in Table 2. Note the little-endian notation! Request ID is either 1 or 3 bytes long, depending on the 5 least significant bits of the first byte of the request ID – if these bits are all set, the command ID extends to 3 bytes.

**Table 2. Request ID fields.**

Bit field	Location (bits)	Values	Comments
Block Type Request/Reply	7	0	Requests only
Block Type Read/Write	6	0,1	0–Read, 1–Write
Reserved	5	0	
Data Type	4–0	0–1F	if 1F, extends to 3B

AMSWire requests are not protected by CRC, because of their short life span (i.e. probability of corruption in memory due to upsets is negligible over the duration of experiment) and because of extremely low bit error rate on the AMSWire links (it is measured to be less than  $10^{-14}$ ). On the contrary, all replies are CRC-protected. Over AMS life span it is expected that some  $10^4$  physics events will be corrupted due to upsets in various memories used in AMS. Those events can be easily identified on the ground with CRC check.

## **6.2 0-length AMSWire packets**

0-length AMSWire packets can represent special types of requests and replies. The type is defined by the Block Control bits BC1 and BC2. Definition and meaning of the 0-length requests and replies are outlined in Tables 3 and 4. Destination of the 0-length request is defined by the commanding context (i.e. addressing path of preceding non-0 length request). If context is not previously defined, receiving node replies.

**Table 3. 0-length AMSWire request packets.**

Request	BC1	BC2	Possible replies
NEXT	0	0	DATA, ABORT
–	0	1	ABORT
ABORT	1	0	END, ABORT
–	1	1	ABORT

**Table 4.0-length AMSWire reply packets.**

Reply	BC1	BC2	Meaning
NEXT	0	0	Ready for the next packet
ERROR	0	1	Request is not understood
ABORT	1	0	Request can not be served
END	1	1	Acknowledgment

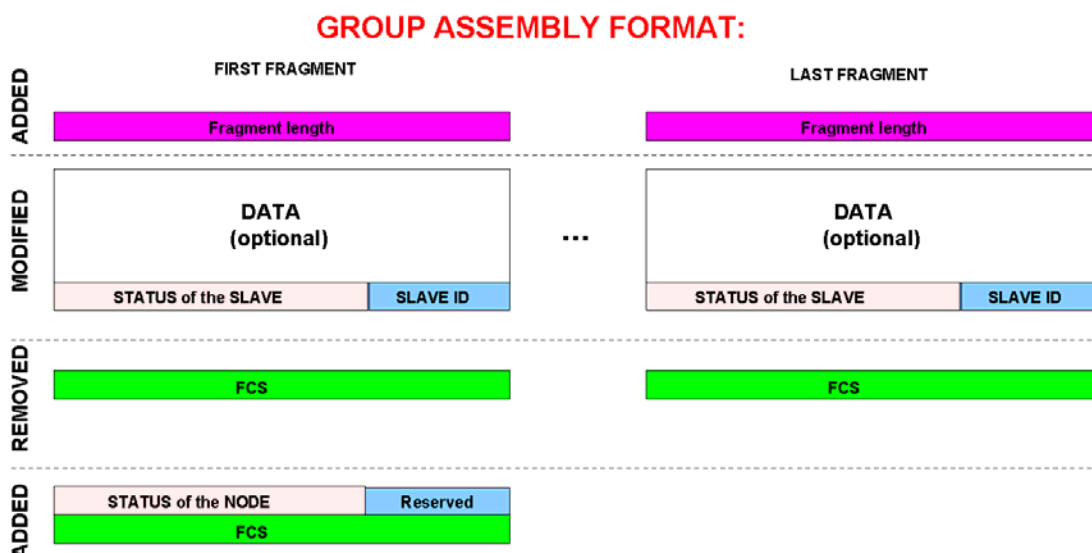
### 6.3 Reply formats

Reply to an AMSWire command may (non-zero length replies) or may not (zero length replies) contain data words. For non-zero length replies their structure is shown in Figure 9. Reply to an AMSWire command contains no header. Non-zero length replies always contain two words at the end of the reply body: reply status of the corresponding node and the CRC Frame Check Sequence. In case of the group command assembly, the reserved part of the reply status word in Figure 9 is filled by the master with the corresponding slave ID. This equally applies to the event building procedure in JINF and JINJ nodes. In case of a command addressed to a slave, the intermediate node does not modify the slave reply.

Assembly of the data replies to a group command is depicted in Figure 10. This is common to all group commands. Individual replies are CRC-checked to verify validity of a slave reply. In case the reply fails CRC check the corresponding error code is set in the status word. The fragment length is pre-pended to every slave reply during the assembly. New FSC for the assembled reply is calculated when copying that fragment to the Group Command build buffer – this FCS is appended to protect the assembled reply at the last step of the building process. Maximal length of assembled reply is limited to 24 KB. In case a fragment which is being assembled makes the total length greater than 24 KB, it is truncated to 4 words (length, RX status, first word of the fragment and status) with the corresponding assembly error code set in the status word as described in Section 6.4.

In the event of all slaves delivering the very same 0-length reply this reply is send to the master as a reply to the group command. In case different reply types from the slaves (for

instance data replies and 0-length replies or different 0-length replies), 0-length replies ABORT, ERROR and NEXT are assembled as 2-word fragments: length and status (with the corresponding reply code set in the slave status word as described in Section 6.4).



**Figure 10** Format of a data reply to Group Command.

#### 6.4 Group Command assembly

When executing group command a temporary list of specified slaves is created. The AMSWire block from the group command is sent to each slave from this list only once. The slave is expected to reply within 0.6s. Passed that delay the processing node will treat such a slave as non-replying and will not repeat the request. Request is not repeated also in case of AMSWire errors detected by the receiver. Assembling node analyses the reply and assigns a unique reply code to every slave from the list. Reply code is stored in the 5 MSbits of the slave status word. Format of the slave status word is shown in Table 5. The most significant bit indicates whether the slave replied with data or not (i.e. for 0-length replies as well as for reply timeout or reply AMSWire errors this bit is set to 0). If the Reply Status word is successfully received the Slave Status field (REPLY STAT\_S in Figure 10) is not modified (see Section 7.2), otherwise this field is left blank. Possible Reply Codes are defined in the Table 6.

**Table 5. Slave status word fields.**

bits	f	e	d	c	b	a	9	8	7	6	5	4	3	2	1	0
field	DAT	Reply Code				Slave Status						Slave ID				

**Table 6. Slave Reply Codes for Group Command assembly.**

Reply Code	Slave Status
DAT bit set to 0	
0x1	0-length reply NEXT
0x2	0-length reply ABORT
0x3	0-length reply ERROR
0x4	0-length reply END
0x5	Timeout
0x6	AMSWire errors
DAT bit set to 1	
0x0	No errors during assembly
0x1	First received packet have no BC1 bit set
0x2	No room to append BC=11 packet
0x3	No valid data
0x4	Reserved for EVBLD (event No mismatch)
0x5	BC=11 packet with CRC error
0x6	Timeout on the reply to NEXT command
0x7	0-length reply to NEXT command
0x8	Intermediate/last packet has wrong BC bit settings
0x9	No buffers available for multi-packet transaction
0xA	No room to append BC=00/10 packet
0xB	Reserved for EVBLD (event No mismatch, multi-packet)
0xC	Multi-packet reply with CRC error

Reply from each slave is appended to the Group Reply using the following rules:

- for 0-length reply or timeout – two words are added to the Group Command reply: reply length (=1) and slave status word;
- for reply received with AMSWire errors – four words are added to the Group Command reply: reply length (=3), content of RX status register, content of RX address register and slave status word;
- for data reply that generate an error during reply processing – five words are added to the Group Command reply: reply length (=4), content of RX status register, reply length, first data word of the reply and slave status word.
- for data reply which does not generate errors during assembly, this reply is added to the Group Command reply assembly according to the scheme shown in Figure 10.

In case of all slaves replying the very same type of 0-length reply, this reply is transmitted to the master without modification. In case of data reply, the assembling node does not set assembly error bit (bit #9) in the status word only if all slaves replied with data fragments that did not generate errors during assembly. Otherwise this bit is set.

## 7 Processing of Commands

The list of implemented AMSWire commands is given in Table 7. Reply to a WRITE command is always 0-length, whereas reply to a READ command may also contain data. In case of errors while receiving a command, or internal errors during execution of the command 0-length ERROR reply is delivered. In case command received without errors, but is inconsistent with the internal list of predefined commands, 0-length ABORT reply is delivered. WRITE command executed successfully is acknowledged with 0-length END reply. Non-0 length replies are formatted according to the Figure 9.

Format of Reply Status word is described in Subsection 7.14. It is important to note that field [bits 10-5] of this word is filled by the replying node, whereas fields [bits 15-11] and [bits 4-0] are filled by the assembling node that executes group command or event building.

### 7.1 Boot Command

This is a special command (WRITE type) as it is primarily handled by the Gate Array and not by the DSP program. It may have one or no parameters. If the parameter is present, it is a header of the valid Program File to be loaded. In case the Boot command is identified, the Gate Array resets the DSP and sets the boot bit in the corresponding RX status register. DSP reset causes loading ROM Monitor, which subsequently analyses the boot command parameters and takes an appropriate action: replies to the boot command and then either loads the program specified in the Boot Command and passes control to it or continues running as ROM Monitor. In case of no parameters, invalid parameter or RX errors, no program is loaded by ROM Monitor. Reply to the Boot Command is 0-length END, ABORT or ERROR.

### 7.2 Read Node Status

Read Node Status Command (READ type) has no parameters. Reply to this command is either non-0 length DATA Block containing information on the Node Status or 0-length ERROR or ABORT. Node Status reply contains 12 words (Table 8):

1. Program version ID. This is a release date: bits 15-12 contain the year (0-15); bits 11-8 contain the month (1-12); and bits 7-0 contain the date (1-31);
2. Program attributes. The following attribute fields are defined:
  - Bits 15-12 – Program Type: values 0–ROM and 1–DAQ;
  - Bits 11-8 – Node Type: values 1–CDP and 2–CDDC;
  - Bits 7-4 – Sub-Detector specifics: values 0, 1, 2, 3, 4, 5, 6 and 7 correspond to no specifics (JINJ only), Tracker, TRD, RICH, ECAL, TOF, LV1 and ECAL Trigger nodes, respectively;
  - Bits 3-0 – Number of AMSWire ports (to the node masters).

• **Table 7. List of AMSWire Commands.**

Data Type			Response time	Comments
Mnemonics	R/W	ID		
DSP Monitoring and Control Commands				
Boot	W	0x40	13ms	any node
Read Node Status	R	0x0C	10 $\mu$ s	any node
Ping	R	0x0D	10-200 $\mu$ s	any node
DSP Power Down	W	0x4E	10 $\mu$ s	any node
Program Test	W	0x55	400 $\mu$ s	any node
FLASH File Read	R	0x05	1-2ms	any node
FLASH File Write	W	0x45	70ms(4s)	any node
FLASH File Test	R	0x06	1-2ms	any node
FLASH File Load	W	0x46	1-2ms	any node
FLASH Summary Read	R	0x07	10 $\mu$ s	any node
FLASH File/Sector Erase	W	0x47	60 $\mu$ s(15s)	any node
File Attribute Set	W	0x48	20 $\mu$ s	any node
Perform I/O	R	0x0F	10-200 $\mu$ s	any node
PM Read	R	0x10	10-200 $\mu$ s	any node
PM Write	W	0x50	10-200 $\mu$ s	any node
DM Read	R	0x11	10-200 $\mu$ s	any node
DM Write	W	0x51	10-200 $\mu$ s	any node
DAQ Monitoring and Control Commands				
Physics Event Read	R	0x01	10-200 $\mu$ s	not ROM
Read Last Evevnt No	R	0x02	10 $\mu$ s	not ROM
Reset Event FIFO	W	0x42	10 $\mu$ s	not ROM
Read HK Info	R	0x03	10-200 $\mu$ s	not ROM
Configuration Read	R	0x09	10-100 $\mu$ s	not ROM
Configuration Write	W	0x49	10 $\mu$ s-500ms	not ROM
Processing Mode Read	R	0x12	10 $\mu$ s	not ROM
Processing Mode Set	W	0x52	10 $\mu$ s	not ROM
Calibration Status	R	0x13	10-400 $\mu$ s	not ROM, CDP only
Calibration Control	W	0x53	10 $\mu$ s	not ROM, CDP only
SD Procedure Status	R	0x14	10-400 $\mu$ s	not ROM
SD Prosedure Control	W	0x54	10 $\mu$ s-500ms	not ROM
Slave Test Status	R	0x16	10 $\mu$ s	not ROM, not CDP
Slave Test Control	W	0x56	10 $\mu$ s	not ROM, not CDP
Slave Mask Read	R	0x17	10 $\mu$ s	not ROM, not CDP
Slave Mask Write	W	0x57	10 $\mu$ s	not ROM, not CDP
BUSY Mask Read	R	0x18	10 $\mu$ s	JINF
BUSY Mask Write	W	0x58	10 $\mu$ s	JINF
DELAY value Read	R	0x19	10 $\mu$ s	JINF
DELAY Value Write	W	0x59	10 $\mu$ s	JINF
BUSY Status Read	R	0x0A	10 $\mu$ s	JINF
BUSY Errors Read	R	0x0B	10 $\mu$ s	JINF
SSF Status	R	0x1A	10 $\mu$ s	JINF
SSF Control	W	0x5A	10 $\mu$ s	JINF
Slow Control and Monitoring Commands				
QLIST Entry Delete	W	0x5B	10 $\mu$ s	not ROM
QLIST Read	R	0x1C	10 $\mu$ s	not ROM
QLIST Write	W	0x5C	10-50 $\mu$ s	not ROM
LeCroy Bus Read	R	0x1D	(150·N) $\mu$ s	not ROM
LeCroy Bus Write	W	0x5D	300 $\mu$ s	not ROM

**Table 8. Format of the reply to Status Command.**

	f	e	d	c	b	a	9	8	7	6	5	4	3	2	1	0
1	Program Version ID															
2	Program Attributes															
3	Subdetector Version ID															
4	Local Node Time (in 10ms ticks)															
5																
6	Node status word															
7	Last Event Number															
8	Number of BUILD errors															
9	Number of AMSW errors															
9	Number of FLASH errors															
9	Reply Status															
12	Frame Check Sequence															

3. Sub-Detector version ID – version release date (same notation as for Version ID);
4. Node Time: 16 LSbits of time (in 10 ms ticks) elapsed since the beginning of the program execution;
5. Node Time: 16 MSbits of elapsed time;
6. Node Status word: bits in the status word are set if the following condition occur:
  - bit 14 – Event Building errors;
  - bit 13 – AMSWire errors;
  - bit 12 – FLASH errors;
  - bit 11 – Buffer Memory Test errors detected at Node Init;
  - bit 10 – Program Memory test errors detected at Node Init;
  - bit 9 – Data Memory test errors detected at Node Init;
  - bit 8 – Program self-test errors;
  - bits 1-0 – AMSWire link ID;
7. Last Event Number assigned;
8. Number of Event building errors;
9. Number of AMSWire errors;
10. Number of FLASH errors;
11. Reply status. Bits 10-6 are set by the replying node, other bits set by its master:
  - bit 15 – DATA bit, set by master when assembling group reply;
  - bits 14-11 – reply code (see Section 6.4), set by master;
  - bit 10 – build conditions error (Sub-Detector specific);
  - bit 9 – build errors (sequencer errors (CDP) or event assembly (CDDC));
  - bit 8 – cumulative node status (OR of bits 11-8 in the Node Status word);
  - bit 7 –COMPRESSED event building mode;
  - bit 6 –RAW event building mode;
  - bit 5 – internal structure bit (0 means group assembly, 1 – no sub-structure);
  - bits 4-0 – slave ID, set by master;
12. Frame Check Sequence.

### **7.3 Ping Command**

Ping Command (READ type) may have from 0 up to 8000 arbitrary parameters. Reply to this command is either non-0 length DATA Block containing the Ping Command parameters with Reply Status and FCS appended or 0-length ERROR or ABORT.

### **7.4 DSP Power Down Command**

DSP Power Down Command (WRITE type) has no parameters. It forces DSP to enter the state with minimal power consumption. Only AMSWire receivers are functional in this mode. To exit POWER DOWN State Boot Command must be sent. Reply to this command is 0-length END or ERROR or ABORT. In case of Boot Command with parameter, memory failure occurs and corresponding bit in RX Status register is set leading to ABORT of the command by the software. That does not happen for Boot command without parameter.

### **7.5 Program Test**

Program TEST Command (WRITE type) has no parameters. A dedicated procedure runs a program self-test and set corresponding bit in the Node status according to the test results. As a result of the test the Node Status is updated. Reply to this command is 0-length END or ERROR or ABORT.

### **7.6 FLASH File Read**

FLASH File Read Command (READ type) has one parameter – the header of the file to be read. Reply to this command is either non-0 length DATA Block containing the requested file (the structure is shown in Figure 8) or 0-length END (in case the requested file does not exist) or ERROR or ABORT.

### **7.7 FLASH File Write**

FLASH File Write Command (WRITE type) contains the File as its DATA body. The file structure is defined in Figure 8. If Write FLASH File Block is sent as several packets, reply to the First packet (unless it is also Last packet) is 0-length NEXT or ERROR or ABORT. Reply to the Intermediate packet is 0-length NEXT or ERROR. Reply to the Last packet is 0-length END or ERROR or ABORT.

## **7.8 FLASH File Test**

FLASH File Test Command (READ type) has one parameter – the header of the file to be tested. Reply to this command is either non-0 length DATA Block containing the file status or 0-length ERROR or ABORT.

## **7.9 Load FLASH File**

FLASH File Load Command (WRITE type) has one parameter – the header of the file to be loaded. If the loaded file contains a Program, control will be passed to it. If the loaded file contains data segments for Q-list or Configuration Procedure the corresponding update procedures will be executed after the file load is successfully completed. Reply to this command is 0-length END or ERROR or ABORT.

## **7.10 FLASH summary Read**

FLASH Summary Read Command (READ type) has no parameters. Reply to this command is either non-0 length DATA Block containing the status of 8 unprotected FLASH sectors or 0-length ERROR or ABORT. A content of every sector is characterized by 4 words:

1. First word at the start address of the FLASH sector (File Header for files).
2. Sector address – BMPAGE value as appears in the BDMA Control register.
3. Sector status: bit 15 is set if sector is bad (either corrupted file or a sector containing bits set to 0); bit 14 is set if sector is being erased.
4. Reserved word.

## **7.11 FLASH File/Sector Erase**

FLASH File/Sector Erase Command (WRITE type) has one parameter – the header of the file to be erased or sector address as it appears in the FLASH Summary. Reply to this command is 0-length END (in case sector erasing started) or ERROR or ABORT.

## **7.12 File Attribute Set**

FILE Attribute Set Command (WRITE type) has two parameters – the header of the file which DEFAULT attribute is to be set and the attribute value (1 -set file to be default, 0 -set file to be not default). Reply to this command is 0-length END or ERROR or ABORT.

### **7.13 Perform IO Command**

Perform IO Command (READ type) contains the IO segment (see Section 5 for description of IO segment) as its DATA body. Reply to this command is either non-0 length DATA Block containing modified IO segment (only for IO read operations) or 0-length ERROR or ABORT.

### **7.14 Program Memory Read**

Program Memory Read Command (READ type) contains the PM segment header (see Figure 8 for description of PM segment header) as its DATA body. Reply to this command is either non-0 length DATA Block containing corresponding PM segment or 0-length ERROR or ABORT.

### **7.15 Program Memory Write**

Program Memory Write Command (WRITE type) contains the PM segment (see Figure 8 for description of PM segment) as its DATA body. Reply to this command is 0-length END or ERROR or ABORT.

### **7.16 Data Memory Read**

Data Memory Read Command (READ type) contains the DM segment header (see Figure 8 for description of DM segment header) as its DATA body. Reply to this command is either non-0 length DATA Block containing corresponding DM segment or 0-length ERROR or ABORT.

### **7.17 Data Memory Write**

Data Memory Write Command (WRITE type) contains the DM segment (see Figure 8 for description of DM segment) as its DATA body. Reply to this command is 0-length END or ERROR or ABORT.

### **7.18 Read Event**

Read Event Command (READ type) has no parameters. Reply to this command is either non-0 length DATA Block containing the physics event fragment (if it is available) or 0-length END (if event is not available) or ERROR or ABORT.

### 7.19 Read Last Event Number

Read Last Event Number Command (READ type) has no parameters. Reply to this command is either non-0 length DATA Block containing Last Assigned Event Number and average event processing time (in 20µs/CDP or 40µs/CDDC ticks) or 0-length ERROR or ABORT. Average processing time is calculated over all collected events using the following formula:

$$\langle t_{n+1} \rangle = 0.5 t_{n+1} + 0.5 \langle t_n \rangle$$

### 7.20 Reset Event FIFO

Reset Event FIFO Command (WRITE type) has no parameters. This is a request to clear all event FIFOs (raw and built events) and set Last Assigned Event Number to zero. Reply to this command is either 0-length END or ERROR or ABORT.

### 7.21 Read Housekeeping Info

Read Housekeeping Info Command (READ type) has no parameters. This command is issued to every node during data taking, it allows monitoring of settings and prompt detection of changes. Reply to this command is either non-0 length DATA Block containing housekeeping information as defined by the sub-detector specific routine (sEVBLD INFO) or 0-length ERROR or ABORT. Format of the reply to this command is shown in Table 9.

**Table 9. Format of the reply to Housekeeping Info Command.**

	f	e	d	c	b	a	9	8	7	6	5	4	3	2	1	0
	<b>Common part</b>															
1	Program Version ID															
2	Subdetector Version ID															
3	Node status word															
4	Last Event Number															
5	Last Event processing time															
	<b>CDP specific words</b>															
6	Calibration Type															
7	Calibration Status															
...	Detector specific hardware Status															
L-1	Reply Status															
L	Frame Check Sequence															
	<b>JINF specific words</b>															
6	BUSY Error_H															
7	BUSY Error_L															
...	Detector specific hardware Status															
L-1	Reply Status															
L	Frame Check Sequence															

### **7.22 Node Configuration Read Command**

Node Configuration Read Command (READ type) has two or more parameters. The first parameter specifies Sub-Detector ID and number of parameters to be read. The following parameters constitute a list of parameter ID to be read-out (detailed description is in Section 8). Reply to this command is either non-0 length DATA Block containing corresponding parameters IDs and values or 0-length ERROR or ABORT.

### **7.23 Node Configuration Write Command**

Node Configuration Write Command (WRITE type) has three or more parameters. The first parameter specifies Sub-Detector ID and number of parameters to be written. The following parameters constitute a list of pairs: parameter ID to write and its value (detailed description is in Section 8). Reply to this command is either 0 length END or ERROR or ABORT.

### **7.24 Processing Mode Read Command**

Processing Mode Read Command (READ type) has no parameters. Reply to this command is either non-0 length DATA Block containing Event Processing Mode value or 0-length ERROR or ABORT.

### **7.25 Processing Mode Write Command**

Processing Mode Write Command (WRITE type) has one parameter – Event Processing Mode value. Bit 0 corresponds to RAW event processing request and bit 1 – to COMPRESSED event processing request. Both bits may be set or unset. Default settings are: 2 for CDP nodes and 0 for CDDC nodes. Reply to this command is 0-length END or ERROR or ABORT.

### **7.26 Calibration Status Command**

Calibration Status Command (READ type) has one parameter. Its value instructs DSP to either read calibration status (value=0) or read-out calibration results (value=1) (details are discussed in Section 9). Reply to this command is either non-0 length DATA Block containing Calibration Status or Sub-Detector dependents Results or 0-length ERROR or ABORT.

### **7.27 Calibration Control Command**

Calibration Control Command (WRITE type) has one or two parameters. The first parameter

specifies the requested action: 0 – start calibration (one more parameter follows – calibration type); 1 – Stop Calibration (no other parameters); 2 – create FLASH File with calibration results (one more parameter follows – File Name) (details are discussed in Section 9). Reply to this command is 0-length END or ERROR or ABORT.

### **7.28 Sub-Detector Procedure Status**

Sub-Detector Procedure Status Command (READ type) has any number of parameters (depending on implementation – details are in Section 10). Reply to this command is either non-0 length DATA Block containing status or results of the implemented Sub-Detector procedures or 0-length ERROR or ABORT.

### **7.29 Sub-Detector Procedure Control**

Sub-Detector Procedure Control Command (WRITE type) has any number of parameters (depending on implementation – details are in Section 10). Reply to this command is 0-length END or ERROR or ABORT.

### **7.30 Slave Test Status**

Slave Test Status Command (READ type) has no parameters. Reply to this command is either non-0 length DATA Block containing 24 pairs of words (according to the maximal number of slaves) or 0-length ERROR or ABORT. Each pair contains a mask word (16 masks for a given slave with the MASK 0 corresponding to bit 0) and slave status word. CDP nodes will nominally reply 0-length ABORT.

### **7.31 Slave Test Control**

Slave TEST Control Command (WRITE type) has no parameters. A dedicated procedure sends Read Status Command to every slave and checks validity of replies. The default mask (MASK 0) is defined according to the valid replies. Reply to this command is 0-length END or ERROR or ABORT. CDP nodes will nominally reply 0-length ABORT.

### **7.32 Slave Mask Read**

Read Slave Mask Command (READ type) has no parameters. Reply to this command is either non-0 length DATA Block containing 24 words (according to the maximal number of slaves) or 0-length ERROR or ABORT. Each word contains a slave mask word – 16 masks for a given slave with the MASK 0 corresponding to bit 0. CDP nodes will nominally reply 0-length ABORT.

### **7.33 Slave Mask Write**

Slave Mask Write Command (WRITE type) has two parameters. First parameter contains two fields: bits 11-8 contain Slave Mask ID (0-15) and bits 7-0 contain 8 MSbits of the mask. Second parameter contains 16 LSbits of the mask. The LSbit of the second word corresponds to Slave 0. Reply to this command is 0-length END or ERROR or ABORT. CDP nodes will nominally reply 0-length ABORT.

### **7.34 BUSY Mask Read**

BUSY Mask Read Command (READ type) has no parameters. Reply to this command is either non-0 length DATA Block containing 2 words or 0-length ERROR or ABORT. Bits 7-0 of the first word contain 8 MSbits of the mask corresponding to CDPs 23-16. Second word contains 16 LSbits of the mask corresponding to CDPs 15-0. Non-JINF nodes will nominally reply 0-length ABORT.

### **7.35 BUSY Mask Write**

Slave Mask Write Command (WRITE type) has zero or two parameters. If no parameters, BUSY mask is calculated from default DAQ mask. In case of two parameters - bits 7-0 of the first word contain 8 MSbits of the mask corresponding to CDPs 23-16. Second word contains 16 LSbits of the mask corresponding to CDPs 15-0. Reply to this command is 0-length END or ERROR or ABORT. Non-JINF nodes will nominally reply 0-length ABORT.

### **7.36 DELAY value Read**

DELAY Value Read Command (READ type) has no parameters. Reply to this command is either non-0 length DATA Block containing 1 words (DELAY value) or 0-length ERROR or ABORT. Non-JINF nodes will nominally reply 0-length ABORT.

### **7.37 DELAY value Write**

DELAY Value Write Command (WRITE type) has one parameter – delay value. Reply to this command is 0-length END or ERROR or ABORT. Non-JINF nodes will nominally reply 0-length ABORT.

### **7.38 BUSY Status Read**

BUSY Status Read Command (READ type) has no parameters. Reply to this command is

either non-0 length DATA Block containing 2 words or 0-length ERROR or ABORT. In case of non-zero length reply, bit 15 of the first word corresponds to bitwise OR of the BUSY Mask and BUSY Status. Bits 7-0 of the first word contain BUSY status of CDPs 23-16. Second word contains BUSY status of CDPs 15-0. BUSY status corresponds to a snapshot at the time of reading. Non-JINF nodes will nominally reply 0-length ABORT.

### **7.39 BUSY Errors Read**

BUSY Errors Read Command (READ type) has no parameters. Reply to this command is either non-0 length DATA Block containing 2 words or 0-length ERROR or ABORT. Bits 7-0 of the first word contain cumulative BUSY errors of CDPs 23-16. Second word contains cumulative BUSY errors of CDPs 15-0. Errors are set according to Figure 2. These two words correspond to the READ&CLEAR registers. Non-JINF nodes will nominally reply 0-length ABORT.

### **7.40 SSF Status Command**

Read SSF Status Command (READ type) has no parameters. Reply to this command is either non-0 length DATA Block containing 1 word (SSF Status: bit0 corresponds to SSF setting; and bit2/3 – to the deviation of the current/latched(R&C) status from the setting.) or 0-length ERROR or ABORT. Non-JINF nodes will nominally reply 0-length ABORT.

### **7.41 SSF Set Command**

SSF Set Command (WRITE type) has one parameter. This parameter takes two values: 0 (OFF) and 1 (ON). Reply to this command is 0-length END or ERROR or ABORT. Non-JINF nodes will nominally reply 0-length ABORT.

### **7.42 QLIST Entry Delete**

QLIST Entry Delete Command (WRITE type) may have one or no parameters. If there is no parameter, this command deletes all expired entries from the Q-List. If parameter is present, it defines the entry ID for which the number of repetitions will be cleared. If parameter value is 0xFFFF, entire Q-List is deleted. Reply to this command is 0-length END or ERROR or ABORT.

### **7.43 QLIST Read Command**

QLIST Read Command (READ type) has no parameters. Reply to this command is either non-0 length DATA Block containing the entire Q-List or 0-length ERROR or ABORT.

#### **7.44 QLIST Write**

QLIST Write Command (WRITE type) has a DM Segment as its DATA body (DM segment structure is illustrated in Figure 8). The DM start address must be 0x2000 on DM Page 0. The data part of the segment contains variable number of words. The first one defines the number of predefined commands which have to be added to the Q-List. Each predefined command consist of 5 data words:

1. Command Header.
2. Number of Repetitions.
3. Repetitions Frequency.
4. First Parameter, W0.
5. Second Parameter, W1.

Reply to this command is 0-length END or ERROR or ABORT.

#### **7.45 LeCroy Bus Read**

LeCroy Bus Read Command (READ type) has  $1+4\cdot N$  parameters (N -number of transactions):

0. Number of requested transactions;
1. Time delay with respect to previous transaction (in 10ms units);
2. LeCroy Bus number is in the field of bits 6-4, other bits are reserved.
3. Data word W0.
4. Data word W1.
5. Repetition of steps [1]-[4] according to the number of transactions.

Reply to this command is either non-0 length DATA Block containing 7 words or 0-length ERROR or ABORT. In case of the non-0 length reply the format is the following:

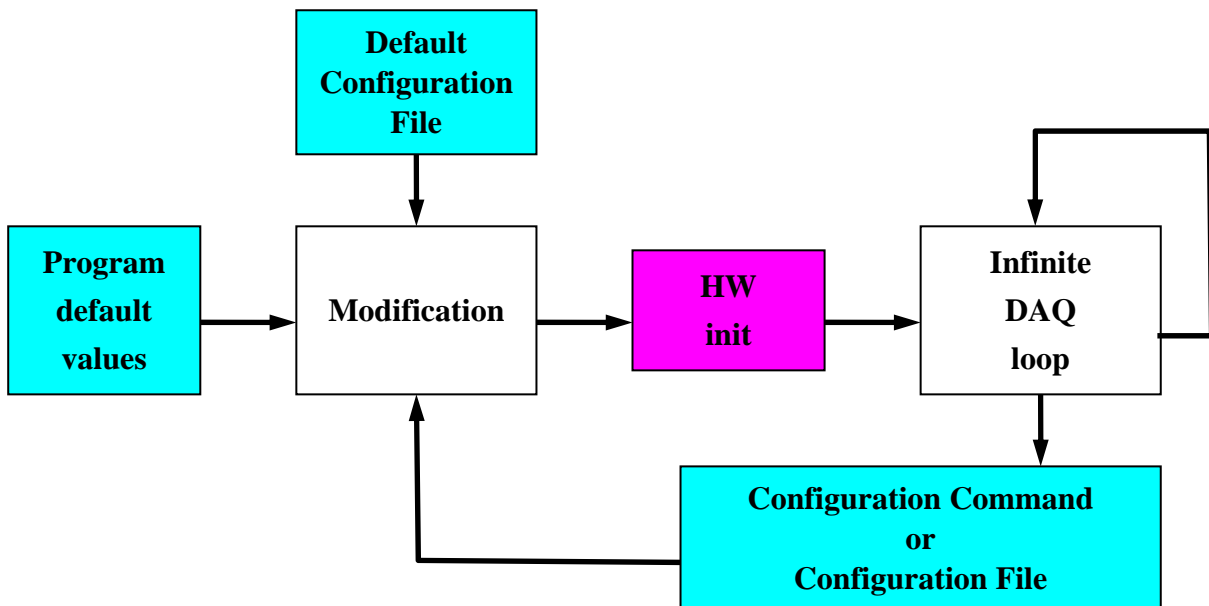
0. Number of requested transactions;
1. Time delay ;
2. LeCroy Bus number.;
3. Data word W0;
4. Data word W1;
5. Data word R0;
6. Data word R1;
7. Repetition of steps [1]-[6] according to the number of transactions.
8. Reply status;
9. Frame Check Sequence.

## 7.46 LeCroy Bus Write

LeCroy Bus Write Command (WRITE type) has one parameter – LeCroy Bus number packed in the field of bits 6-4. This command is used to initialize corresponding LeCroy Bus. Reply to this command is 0-length END or ERROR or ABORT.

## 8 Node Initialization and Configuration

A set of configurable parameters is defined for every sub-detector. Default value for each parameter is assigned during the initialization of the DAQ program. It may be overwritten by another value from default Configuration File during initialization. It also may be modified later using Configuration File or sending an AMSWire command (Section 7.23). A block diagram in Figure 11 illustrates configuration process in CDP nodes.



**Figure 11** Block diagram of Configuration Procedure.

Some parameters may be loaded directly to the corresponding hardware, other may be loaded only as a set using complex hardware initialization procedures (TDC initialization in S-crate for instance). Therefore the latest set of configuration parameters is kept in the DSP Data memory throughout the lifespan of DAQ Program (location – DM page 4, 0x0000–0x1000). This set is updated every time when Configuration File is loaded or AMSWire configuration command is executed. Configuration procedure is implemented in two routines: sSDCONFIG\_RD and sSDCONFIG\_WR. It is up to sub-detector groups to implement specifics of their detectors in these two routines.

## 8.1 Configuration File

Configuration File is a “Data Set” type FLASH file with only one DM segment. Segment Address word of Segment header must be 0x9000. The first word of the segment body is number of configuration parameters; it also contains Sub-detector ID in bits 15-12. Then follow pairs of parameter ID and parameter value. Though the definition of parameter ID depends on the detector specific implementation, it is convenient to have two fields in this word – Group ID and Parameter ID within the Group. The last word is the CRC Frame Check Sequence. The format of the file is shown in Table 10.

**Table 10. Configuration File Format.**

	f	e	d	c	b	a	9	8	7	6	5	4	3	2	1	0	
0	0	1	ATT			FILE NAME							0	0	0	1	
1	0	1	Segment Length (N-2 words)														
2	1	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	
3	SubDet ID				Number of parameters												
...																	
2L	Group				Subgroup				Parameter ID								
2L+1	Parameter value																
...																	
N	FCS																

## 8.2 AMSWire Configuration Command

AMSWire Configuration Command (Sections 7.22 and 7.23) has the very same format as the segment body of the Configuration File. The first parameter in the Configuration Read or Write Command is the number of parameters – word 3 in Table 10. Then follows the list of parameters. Each Parameter ID is defined as the word 2L in Table 10. Configuration Command is not protected by FCS.

## 8.3 Configuration of JINF and JINJ nodes

A set of configurable parameters for JINF consist of 2 or more groups:

1. Group 0 – 32 parameters corresponding to 16 slave masks;
2. Group 1 – 4 parameters corresponding to SSF state, BUSY Mask and S/H time;

3. Other Groups – detector specific configuration parameters for slaves on LeCroy Bus. Any number of groups may be introduced by sub-detectors for initialization of detector-specific hardware on LeCroy buses.

Default value for each parameter is assigned during the initialization of the DAQ program. Any default value may be overwritten by another value from default Configuration File during initialization. It also may be modified later using a Configuration File or sending a corresponding AMSWire command. Whenever a configuration parameter is changed in the hardware, it is read back and compared with the original setting. Result of this comparison is reflected in the status word for the Group. Other Groups are related to detector specifics and may be introduced by sub-detector groups for initialization of detector-specific hardware on LeCroy buses. The breakdown between common framework and detector specifics is the following: all related framework routines are in ..\src\jinf\init.asm and all detector specifics is in the two files - ..\src\xdr\jinf\_sd.asm and ..\src\xdr\jinf\_sd.asm. These detector-specific files correspond to E-crate, they may be used by other sub-detectors as templates.

A set of configurable parameters for JINJ consists only of Slave Masks (Group 0 in the list above). Detector-independent default configuration parameters for JINF/JINJ are listed in Table 11.

**Table 11. JINF configuration parameters**

Group	SubG	ID	Parameter	Default value	Range
<b>Slave Masks</b>					
0	1	0	MASK 0x0, MSByte	0x0000	0x0000-0x00FF
0	1	1	MASK 0x0, 2 LSBytes	0x0000	0x0000-0xFFFF
...					
0	1	30	MASK 0xF, MSByte	0x0000	0x0000-0x00FF
0	1	31	MASK 0xF, 2 LSBytes	0x0000	0x0000-0xFFFF
<b>JINF internal registers</b>					
1	1	0	SSF State	0x0001	0x0000-0x0001
1	1	1	BUSY_H Mask	0x0000	0x0000-0x00FF
1	1	2	BUSY_L Mask	0x0000	0x0000-0xFFFF
1	1	3	S/H Time	0x0080	0x0000-0x00FF
<b>LB Configuration parameters (optional)</b>					
...					

## 9 Calibration Procedure

There are two types of calibration that can be performed for each sub-detector – with internal and with external trigger. Internal Calibration processes events triggered by DSP with the user-defined frequency. The bottleneck of this approach is that each node within one crate initiates digitizing of analog signals asynchronous to other nodes, thus making data

acquisition conditions different from physics event collection. To overcome that external calibration procedure is using a trigger common to all nodes in AMS. This common trigger is provided by the internal JLV1 programmable pulse generator or by JLV1 DSP.

Calibration procedure is started and stopped by the AMSWire commands Calibration Control. Calibration Init command sets the Calibration Request which is then cleared by the Calibration Stop command. As long as the Calibration Request is set all triggered events will be processed by the calibration processing routine and not by the event building routine – these two modes are self exclusive. Each sub-detector calibration routine consumes a predefined number of triggered events (as defined by cCALIB\_NEVT) and makes all the necessary calculations of pedestals, sigmas and masks. Calibration Request remains set until it is cleared by the Calibration Stop command. Therefore if the predefined number of events is already collected, calibration processing routine does not include extra events in the accumulated sums, but only releases events from RAW event buffer.

**Table 12. Calibration Commands Formats.**

	f	e	d	c	b	a	9	8	7	6	5	4	3	2	1	0	
<b>Calibration Control – INIT (call sCALIBR_WR_INIT)</b>																	
1	Pass (0x2e)						Command ID (0x53)										
2	Control Type (0x0000)																
3	Ext	Frequency (in 20 $\mu$ s ticks) – only for Ext=0															
<b>Calibration Control – STOP (call sCALIBR_WR_STOP)</b>																	
1	Pass (0x2e)						Command ID (0x53)										
2	Control Type (0x0001)																
<b>Calibration Control – FILE Create (call sCALIBR_WR_FILE)</b>																	
1	Pass (0x2e)						Command ID (0x53)										
2	Control Type (0x0002)																
3	reserved				File Name								reserved				
<b>Calibration Control – FILE Load</b>																	
1	Pass (0x2e)						Command ID (0x53)										
2	Control Type (0x0003)																
3	File Header																
<b>Calibration Status – Current Calibration Status</b>																	
1	Pass (0x2e)						Command ID (0x13)										
2	Status Type (0x0000)																
<b>Calibration Status – DATA (call sCALIBR_RD_DATA)</b>																	
1	Pass (0x2e)						Command ID (0x13)										
2	Control Type (0x0001)																
<b>Calibration Status – Dynamic Pedestals (call sCALIBR_RD_DPED)</b>																	
1	Pass (0x2e)						Command ID (0x13)										
2	Control Type (0x0002)																

AMSWire Calibration commands together with their parameters are listed in Table 12. Calibration type is defined by Calibration Control INIT command – for internal calibration Ext bit in the second command parameter is set to 0, and for external calibration –

to 1. In the latter case the data from field Frequency is not used. To comply with this calibration scheme, sub-detector groups provide the following Calibration routines (examples are in ..\sdr\cal.asm):

- sCALIBR\_WR\_INIT – to initialize necessary arrays;
- sCALIBR\_WR\_EVENT – to accumulate required sums;
- sCALIBR\_WR\_STOP – to calculate means, sigmas, thresholds, etc..;
- sCALIBR\_WR\_FILE – to make a FLASH file out of calibration results;
- sCALIBR\_RD\_DATA – to provide calibration results (peds, rms, thresh, ...);
- sCALIBR\_RD\_DPED – to provide current values of dynamic pedestals.

**Table 13. Format of the reply to Calibration Status Command.**

	f	e	d	c	b	a	9	8	7	6	5	4	3	2	1	0
1	Ext	Frequency (in 20 $\mu$ s ticks)														
2	R	D	F	C	I	Failure Code										
3	Number of events requested															
4	Number of triggers received															
5	Number of events processed															
6	Calibration Start Time (in 10ms ticks)															
7	Calibration Stop Time (in 10ms ticks)															

Format of the reply to Calibration Status with first parameter equal to 0 (Current Calibration Status) is shown in Table 13. The first word is the Calibration request as interpreted by the Calibration Procedure. The second word is current calibration status – bit R set means calibration procedure is running; bit D set means calibration results are available; bit F means there was a failure during the calibration running; bit C set means calibration procedure collected enough data and ready for final processing and bit I means calibration parameters are loaded from a file. The following failure codes are defined:

- 1 – sequencer memory failure
- 2 – event length problem
- 3 – event number problem
- 4 – not enough data
- 5 – warning (S/D dependent)

### 9.1 Internal Calibration

For Internal Calibration a programmable timer is set by DSP according to the corresponding parameter of the Calibration Init command. Upon expiration of this timer DSP will start sequencer (if it is not BUSY) and set the timer again. After triggering the predefined number of events (cCALIB\_NEVT) DSP will not start sequencer any more for this calibration run. This allows calibration routine sCALIBR\_WR\_STOP to check whether there were other events (not triggered by DSP) and set the corresponding failure code.

## **9.2 External calibration**

For External Calibration the trigger frequency and the number of triggers are not controlled by the calibration procedure itself – therefore the calibration processing routine consumes all events it needs to accomplish the calibration and discards all events beyond its needs. The status of the calibration can be monitored by a corresponding AMSWire command (see Tables 12 and 13). Frequency of external calibration is defined by Calibration Control command to JLV1 node [10].

## **10 Sub-detector Specific procedures**

Sub-detectors may define more procedures in order to address their specific needs – for instance powering ON and OFF boards inside a crate, initialize/reinitialize some parts of their hardware, performing DAC calibration or have extra debugging capabilities that go beyond those described in this note. This can be achieved with Sub-detector Specific Procedures which should be implemented in the two following routines: `sSDPROC_WR`, `sSDPROC_RD`. These sub-detector specific commands accept any number of parameters.

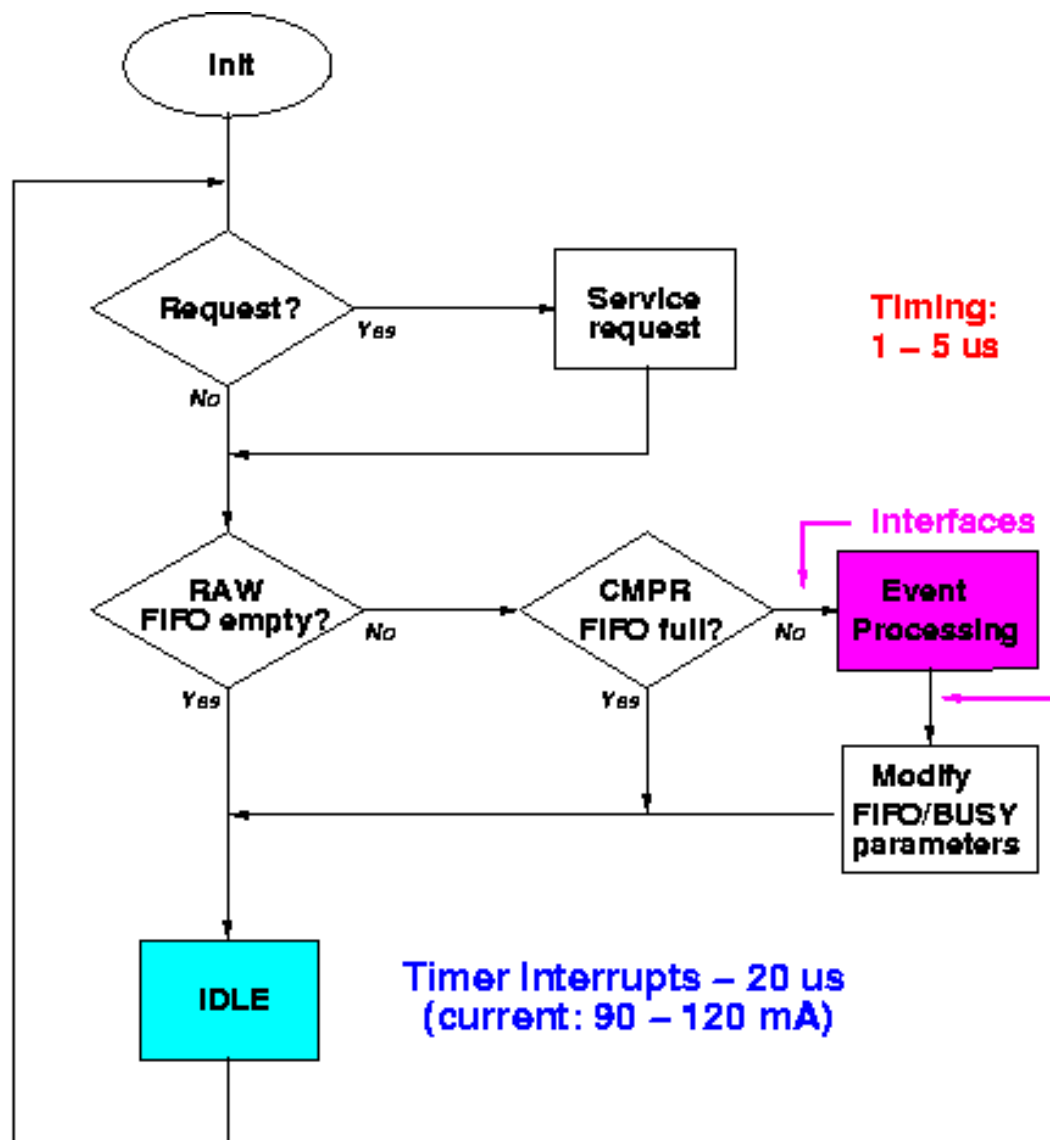
## **11 Event Building Procedure in CDP**

At present there is no Slow Control Procedure in the CDP nodes (with the exception of SDR2). Any information related to Slow Monitoring is retrieved and delivered by the sub-detector specific routine `sEVBLB INFO` which services AMSWire Command Read Housekeeping Info. Program Flow in CDP is presented in Figure 12. IDLE state at the end of the infinite loop in this figure allows to save up to 30 W of power if used in all 240 active CDP nodes.

In response to a LV1 trigger a CDP node sets BUSY signal for the time needed to digitize analog information. The digitized amplitudes (RAW event fragments) are placed to a dedicated area in the Buffer Memory, called raw event FIFO, which has enough room to store up to 4 Event fragments. It is important to note that when raw event FIFO gets full (i.e. 4 events are stored) BUSY is not removed by the CDP node. In that case BUSY is removed only when DSP reads out an event and frees the space in raw event FIFO.

Physics data processing routine (`sEVBLD CMPRS`) is written by each sub-detector group taking into account detector specifics. The calling routine provides read and write addresses in the Buffer Memory. The data processing starts with reading the raw event (`cBM RAWSIZ-1` words) from the I/O register `ioBM DATA RD`. The length of the event is a constant which depends only on the sub-detector – it is defined during the compilation time. The processed

data are written to the Buffer Memory using the I/O register ioBM DATA WR. The processed event format is individual for each sub-detector. The physics data processing routine returns the status (dmBUILD STAT) of processing and the length (in register AR) of the written processed event. The format of status word is described in Section 7.2. Of particular interest to Event Building procedure in CDP are bits 9 and 10 of the status word. Bit 9 is set if there is sequencer memory failure (data part of event is discarded in this case) or there is a mismatch between sequencer event number and internal DSP event number (event is built in this case and the event number corresponds to that of the sequencer). Bit 10 setting is detector-specific and is not altered by the framework software.



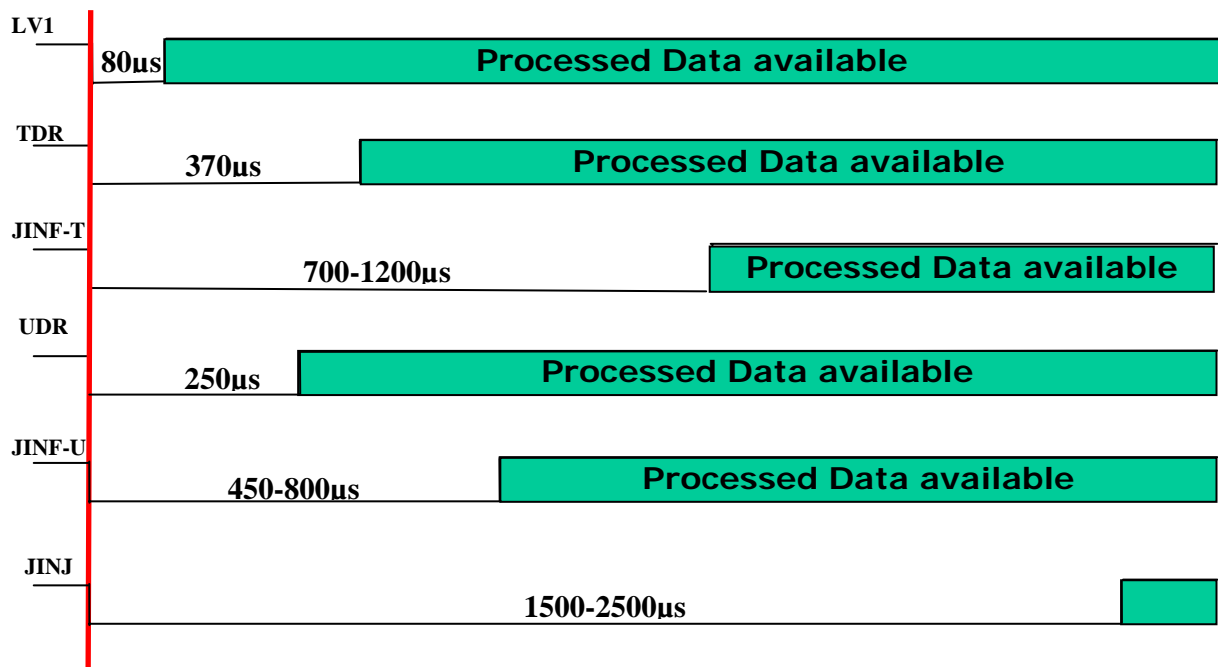
**Figure 12.** Block diagram of CDP Event Building

### 11.1 Event Processing in CDP

TDR processing is described in ref [4]; UDR – in ref [5]; RDR – in ref [6]; EDR and ETRG – in ref [7]; SDR – in ref [8]; and JLV1 – in references [9] and [10].

## 12 Event Building Procedure in CDDC

Program Flow in CDDC is similar to that of CDP. CDDC periodically starts requesting event fragments from its slaves. If none of the slaves replies with a data event, CDDC set a 100  $\mu\text{s}$  timer for the next round of requests – during this 100  $\mu\text{s}$  period it will not request events and will only serve requests from its masters. If at least one slave replied with an event, the request will be repeated to all those slaves that replied with no event in that round. In order to compensate the time lag for availability of the event fragments belonging to the very same physics event in JLV1 and JINF-T (the biggest average time lag) the assembling node will repeat the request to nodes that replied no event during 10ms. On average this time lag is of the order of 1ms (as shown in Figure 13) – this is increased ten-fold to account for possible fluctuations in the sequencer running time, event processing time by DSP and time lag induced by servicing AMSWire requests. In addition, the request to such a slave is sent no less than 3 times per built event. Maximal length of the assembled event is limited to 24 kB.



**Figure 13.** Time diagram of data availability in DSP nodes for the same physics event.

Assembly of the event fragments in CDDC node is very similar to Group Command assembly (Section 6.4). The only exception is the treatment of the very first word in a slave reply. The first word of any event fragment is the event number. Therefore, an event number from each event fragment is compared with the internal event number of the building node. In case of mismatch, the corresponding error code is set in the slave status word (Table 6).

Event processing time is on average 300-400  $\mu$ s at high trigger rates, but highly depends on running conditions. In general event processing time comprise the following bits:

1. AMSWire request latency – time between CDDC starts preparing a request to a slave and the slave starts its AMSW transmitter. Minimal latency is 20  $\mu$ s per slave, but it may be much bigger if the request arrives when the slave is already building an event or serving a request from another master.
2. AMSWire transmission time amounts to about 0.2  $\mu$ s per 16bit word. It should be noted that in normal running conditions two slave links are active at any given moment. Effectively the transmission time (as well as corresponding latency) is reduced by factor 2.
3. Fragment processing time (IO Read/Write operations) is 0.1  $\mu$ s per word.
4. Event building overall overhead amounts to about 10  $\mu$ s.

In order to optimize the AMSWire throughput for event building purposes, the slaves with highest data flows should be connected to different gates in CDDC (i.e. link groups 0-3, 4-7, ..., 20-23).

### 12.1 Event building format in JINJ

Format of an event built by a JINJ is shown in Table 14. Nominally event fragment built by JINJ consists of 5 CDP fragments (SDR2 nodes and JLV1 node) and 14 CDDC fragments assembled following a procedure described in Section 6.4. No further compression is applied.

**Table 14. JINJ event fragment Format.**

	f	e	d	c	b	a	9	8	7	6	5	4	3	2	1	0
<b>0</b>	Event number															
	...															
<b>a0</b>	CDP Event Fragment Length (=n)															
<b>a...</b>	CDP Event Fragment Data															
<b>an</b>	D	Reply Code				Slave Status		Pr Mode		1		Slave ID				
	...															
<b>b0</b>	JINF Event Fragment Length (=k)															
<b>b...</b>	JINF Event Fragment Data															
<b>bk</b>	D	Reply Code				Slave Status		Pr Mode		0		Slave ID				
	...															
<b>N-1</b>	0	0	0	0	0	Slave Status		Pr Mode		0	0	0	0	0	0	0
<b>N</b>	FCS															

## 12.2 Event building format in JINF

Format of an event built by a JINF is shown in Table 15. Depending on the node JINF event fragment data consist of several CDP event fragments. In nominal running conditions majority of CDP event fragments are 1 word long – they carry no physics information and consist only of CDP status word. During event assembly in JINF all one-word long fragments with no errors in the status words are omitted from the event built by JINF. In order to keep track of these CDP fragments a two-word long mask is added (words m1 and m0 in Table 15) with each set bit corresponding to omitted CDP fragment). Any set bit (from 23 to 0) corresponds to an omitted CDP fragment.

**Table 15. JINF event fragment Format.**

	f	e	d	c	b	a	9	8	7	6	5	4	3	2	1	0
<b>0</b>	Event number															
<b>a0</b>	CDP_F Event Fragment Length (=n)															
<b>a...</b>	CDP_F Event Fragment Data															
<b>an</b>	D	Reply Code				Slave Status		Pr Mode		1	Slave ID					
	***															
<b>b0</b>	CDP_L Event Fragment Length (=k)															
<b>b...</b>	CDP_L Event Fragment Data															
<b>bk</b>	D	Reply Code				Slave Status		Pr Mode		1	Slave ID					
<b>m1</b>	Mask for slaves 23-16															
<b>m0</b>	Mask for slaves 15-0															
<b>N-1</b>	0	0	0	0	0	Slave Status		Pr Mode		0	0	0	0	0	0	0
<b>N</b>	FCS															

## 13 Slow Control Procedure

Slow Control Procedure is a list of commands which are executed time-to-time with a relatively low frequency, such that there is almost no interference with the Physics Data Collection. These commands collect and store information on variable system parameters. This information may be retrieved by the node master to ensure that Physics Data were collected in steady conditions. Main sources for this information are: slaves on LeCroy buses connected to JINF as well as IO registers and Data Memory in CDP and CDDC nodes.

Up to 910 fixed-format commands can be stored in Q-List. To minimize the processor time spent on Slow Control Procedure only one command is retrieved from the Q-List on each pass of the infinite loop in Figure 5. Command execution time is checked against the node Base Time and decision is taken on whether to start command execution or not. Each command is executed a specified number of times. After that command is expired, i.e. not executed any more. When command is executed, results are stored and, if command is not yet expired, the execution time is updated according to the command frequency. Execution

frequency ranges from once per 10 ms to once per 46 hours. For execution frequencies on the order of 100 sec the average time spent by DSP in the Slow Control Procedure is 0.9  $\mu$ s. A Q-List Command may have its own execution frequency, or may be executed with a delay with respect to the preceding command in the Q-List. In the latter case these commands form a group with a unique execution frequency for the entire group. A Command Group may contain an arbitrary number of commands. A Q-List command has the following format:

1. Command Header word has the following bit fields:
  - Bits 15-8 – Execution Frequency 8 MSbits;
  - Bit 7 – Execution status (1, if command was executed at least once);
  - Bits 6-4 – Command attribute (Bus Number for LeCroy command, memory type and operation type for other command types);
  - Bit 3 – Command location within the Command Group (First);
  - Bit 2 – Command location within the Command Group (Last);
  - Bits 1-0 – Command Type (00 – LeCroy Command; 01 – AMSWire Command; 10 – Memory Operation; 11 – IO Operation).
2. Number of Repetitions (if First bit is set) or Command Offset within the Group (if First bit not set). For the First Command in a Group this word has the following bit fields:
  - Bit 15 – Execute Forever (if set command is executed forever);
  - Bits 14-0 – Number of repetitions.
3. Execution Frequency 16 LSbits (in 10ms ticks).
4. First Parameter W0, depends on Command Type (LB W0 / AMSW Data Type / Address).
5. Second Parameter W1, depends on Command Type (LB W1 / AMSW Parameter / Value).
6. Execution Time 16 LSbits.
7. Execution Time 16 MSbits.
8. Data Word R0, depends on Command Type (LB R0 / AMSW Status / Address).
9. Data Word R1, depends on Command Type (LB R1 / AMSW Parameter / Value).

Though not limited to LeCroy commands only, the Slow Control Procedure was designed to handle LeCroy in a way that does not compromise the performance of the Event Building. Each transaction on the LeCroy bus takes about 150  $\mu$ s, with most of this time DSP is free for Event Building or servicing AMSWire requests. Therefore, when a Q-List command is ready for execution, DSP only initiates the transaction by starting its SPORT0 TX and then immediately leaves Slow Control Procedure. Intermediate steps of LeCroy transaction are processed by SPORT0 TX/RX interrupt service routines. When transaction is completed the

interrupt service routine sets completion flag. Upon detection of this flag, Slow Control procedure updates QList table. No new Slow Control commands will be retrieved from the Q-List during ongoing LeCroy Bus transaction. Any LeCroy command received by the node during the execution of another LeCroy command from the QList will wait for completion of the ongoing command.

QList may be created from a QList File stored in FLASH memory or by a corresponding AMSWire command. Format of the QList File is shown in Table 16. Format of the AMSWire command corresponds to the segment body of the QList file.

**Table 16. Qlist File Format.**

	f	e	d	c	b	a	9	8	7	6	5	4	3	2	1	0
0	0	1	ATT			FILE NAME							0	0	0	1
1	0	1	Segment length (N-2 words)													
2	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0
3	Number of Qlist Commands															
...																
i	Execution frequency, 8MSbits							0	Attribute			Location		Type		
i+1	F	Number of repetitions / Time offset														
i+2	Execution frequency, 16LSbits															
i+3	First parameter value															
i+4	Second parameter value															
...																
N	FCS															

## 14 Data and Program Memory usage

There are 4 Program Memory pages and 4 Data Memory pages of internal DSP memory. A PM page is organized as 8K of 24bit words and a DM page is organized as 8K of 16bit words. Only one PM and one DM page are accessible to DSP all the time. Other pages are accessible depending on the values of PMOVLAY and DMOVLAY registers. Therefore the following distribution of internal memory resources is currently adopted:

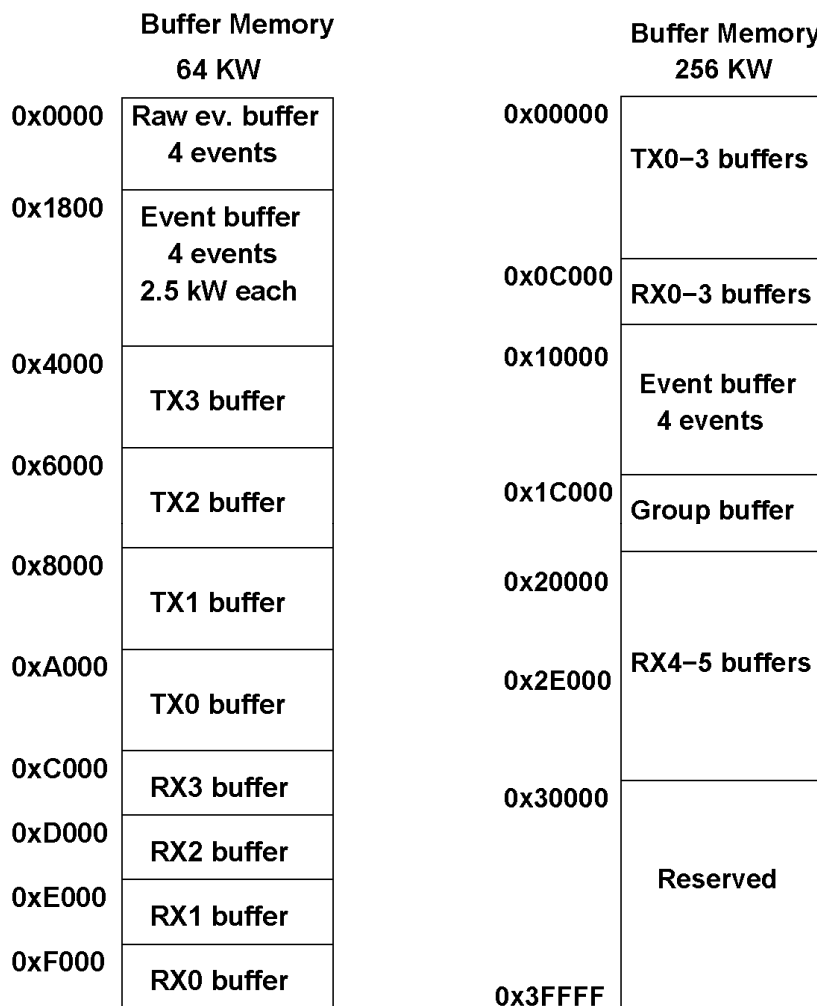
- PM AAC: Main program and constants/variables, address range 0x0000-0x1FFF;
- PM OV0: Buffer for FLASH and AMSWire operations, 0x2000-0x3FFF;
- PM OV4: Event builder program, address range 0x2000-0x3FFF;
- PM OV5: Calibration program, address range 0x2000-0x3FFF;
- DM AAC: Constants and variables, address range 0x2000-0x3FFF. The breakdown is the following: 0x2000-0x31FF for loading/updating Q-lits; 0x3200-0x3DFF for subdetector usage; 0x3E00-0x3ECF - scratch and IRQ temp area for

the framework; 0x3ED0-0x3FDF - framework constants and variables; 0x3FE0-0x3FFF – memory mapped DSP registers;

- DM OV0: Buffer for FLASH and AMSWire operations , 0x0000-0x1FFF. This page is also used by Calibration Procedure – refrain from using FLASH commands during calibration;
- DM OV4: Configuration parameters, address range 0x0000-0x1FFF;
- DM OV5: Q-List, address range 0x0000-0x1FFF.

## 15 Buffer Memory usage

Allocation of the Buffer Memory in CDP (size – 64 KW) and CDDC (size – 256 KW) is shown in Figure 14. Provisions for 4 AMSWire links connecting the node to its masters are made for all nodes, even for those with 2 links only.



**Figure 14.** Buffer Memory organization

## 16 Data Protection

CRC algorithm (16 bit) is used to protect data. Polynomial is  $g = x^{16} + x^{12} + x^5 + x^0$ .

### 16.1 FCS Calculation in Gate Array

Frame Check Sequence (FCS) calculations are done on the fly in the Gate Array. FCS is calculated independently for DSP– read and write operations to the Buffer Memory. The conversion algorithm is given below just for the record. Notations: (c) – FCS value before processing word (d); (d) – data word; (r) – FCS after processing data word (d). Bit ordering: 15 – MSB; 0 – LSB; XOR operation is denoted as  $\wedge$ .

- $r(15) = d(10) \wedge c(10) \wedge d(3) \wedge c(3) \wedge d(7) \wedge c(7) \wedge d(11) \wedge c(11)$ ;
- $r(14) = d(9) \wedge c(9) \wedge d(2) \wedge c(2) \wedge d(6) \wedge c(6) \wedge d(10) \wedge c(10)$ ;
- $r(13) = d(8) \wedge c(8) \wedge d(1) \wedge c(1) \wedge d(5) \wedge c(5) \wedge d(9) \wedge c(9)$ ;
- $r(12) = d(7) \wedge c(7) \wedge d(0) \wedge c(0) \wedge d(4) \wedge c(4) \wedge d(15) \wedge c(15) \wedge d(8) \wedge c(8)$ ;
- $r(11) = d(11) \wedge c(11) \wedge d(15) \wedge c(15) \wedge d(6) \wedge c(6) \wedge d(10) \wedge c(10) \wedge d(14) \wedge c(14)$ ;
- $r(10) = d(10) \wedge c(10) \wedge d(14) \wedge c(14) \wedge d(5) \wedge c(5) \wedge d(9) \wedge c(9) \wedge d(13) \wedge c(13)$ ;
- $r(9) = d(9) \wedge c(9) \wedge d(13) \wedge c(13) \wedge d(4) \wedge c(4) \wedge d(15) \wedge c(15) \wedge d(8) \wedge c(8) \wedge d(12) \wedge c(12)$ ;
- $r(8) = d(8) \wedge c(8) \wedge d(12) \wedge c(12) \wedge d(3) \wedge c(3) \wedge d(14) \wedge c(14) \wedge d(7) \wedge c(7) \wedge d(11) \wedge c(11) \wedge d(15) \wedge c(15)$ ;
- $r(7) = d(7) \wedge c(7) \wedge d(11) \wedge c(11) \wedge d(15) \wedge c(15) \wedge d(2) \wedge c(2) \wedge d(13) \wedge c(13) \wedge d(6) \wedge c(6) \wedge d(10) \wedge c(10) \wedge d(14) \wedge c(14)$ ;
- $r(6) = d(6) \wedge c(6) \wedge d(10) \wedge c(10) \wedge d(14) \wedge c(14) \wedge d(1) \wedge c(1) \wedge d(12) \wedge c(12) \wedge d(5) \wedge c(5) \wedge d(9) \wedge c(9) \wedge d(13) \wedge c(13)$ ;
- $r(5) = d(5) \wedge c(5) \wedge d(9) \wedge c(9) \wedge d(13) \wedge c(13) \wedge d(0) \wedge c(0) \wedge d(11) \wedge c(11) \wedge d(4) \wedge c(4) \wedge d(8) \wedge c(8) \wedge d(12) \wedge c(12)$ ;
- $r(4) = d(4) \wedge c(4) \wedge d(15) \wedge c(15) \wedge d(8) \wedge c(8) \wedge d(12) \wedge c(12)$ ;
- $r(3) = d(3) \wedge c(3) \wedge d(14) \wedge c(14) \wedge d(7) \wedge c(7) \wedge d(11) \wedge c(11) \wedge d(15) \wedge c(15)$ ;
- $r(2) = d(2) \wedge c(2) \wedge d(13) \wedge c(13) \wedge d(6) \wedge c(6) \wedge d(10) \wedge c(10) \wedge d(14) \wedge c(14)$ ;
- $r(1) = d(1) \wedge c(1) \wedge d(12) \wedge c(12) \wedge d(5) \wedge c(5) \wedge d(9) \wedge c(9) \wedge d(13) \wedge c(13)$ ;
- $r(0) = d(0) \wedge c(0) \wedge d(11) \wedge c(11) \wedge d(4) \wedge c(4) \wedge d(8) \wedge c(8) \wedge d(12) \wedge c(12)$ ;

## 16.2 C-code calculation of FCS

A piece of C-code to calculate CRC is shown in Figure 15.

```
#include <stdio.h>
#include <stdlib.h>

int  errno;

unsigned short const g=0x1021;

void main()
{
    unsigned int j;
    unsigned short d, i, k, fcs;
    unsigned short fcs_table[256];
    unsigned char  data[];

    /* prepare lookup table */
    for (i=0; i<256; i++)
    {
        fcs=0x0000;
        d = i<<8;
        for (k=0; k<8; k++)
        {
            if ((fcs^d)&0x8000) fcs = (fcs<<1)^g;
            else fcs <<= 1;
            d <<= 1;
        }
        fcs_table[i]=fcs;
    }

    /* define data */
    ndata = ...
    data[ ] = ....

    /* calculate FCS */
    fcs=0xFFFF;
    for (i=0; i<ndata; i++) {
        fcs = fcs_table[(fcs>>8)^data[i]] ^ (fcs<<8);
    }
    printf ("fcs = %04x\n", fcs);

    return;
}
```

**Figure 15** Example of C-code to calculate FCS

## **17 External AMSWire interfaces**

Two interfaces between PC and AMSWire are available – EPP-AMSWire box and PCI-AMSWire card

## **18 Conversion Program for FLASH File System**

Described in reference [11], updated by F.Pilo ([federico.pilo@pi.infn.it](mailto:federico.pilo@pi.infn.it)), installation program is included in the source code distribution

## **19 Availability of the code and documentation**

Documentation (including all references) as well as the source code are available at <http://ams.cern.ch/AMS/DAQsoft/Welcome.html>

## 20 Appendix

### 20.1 ADSP-2187L Register Set

In addition to registers set defined in the manual [1], the following registers are defined to control the AMSWire RXs and TXs and for accessing Buffer Memory. Table 17 shows I/O WRITE registers and Table 7 – I/O READ registers.

**Table 17. Registers (I/O Write) implemented in FPGA**

Register name	I/O Address			Default value
	xDR	JINF	JINJ	
ioBM_DATA_WR	0x200	0x200	0x200	—
ioPORT0_TXAD_WR	0x600	0x600	0x600	—
ioPORT0_TXST_WR	0x601	0x601	0x601	—
ioPORT1_TXAD_WR	0x604	0x604	0x604	—
ioPORT1_TXST_WR	0x605	0x605	0x605	—
ioPORT2_TXAD_WR	0x610	—	0x610	—
ioPORT2_TXST_WR	0x611	—	0x611	—
ioPORT3_TXAD_WR	0x614	—	0x614	—
ioPORT3_TXST_WR	0x615	—	0x615	—
ioBM_ADWR_WR	0x608	0x608	0x608	—
ioBM_ADRD_WR	0x609	0x609	0x609	—
ioBM_CRCWR_WR	0x60A	0x60A	0x60A	—
ioBM_CRCRD_WR	0x60B	0x60B	0x60B	—
ioBM_ADDREXT_WR	—	0x60F	0x60F	0x00
ioSEQ_STAT_WR	0x60C	—	—	—
ioSEQ_DATA_WR	0x60D	—	—	—
ioTOF_ADDR_WR	0x60E	—	—	—
ioTOF_DATA_WR	0x60F	—	—	—
ioSSF_WR	—	0x60E	—	—
ioMASKH_WR	—	0x610	—	—
ioMASKL_WR	—	0x611	—	—
ioDELAY_WR	—	0x616	—	—
ioPORT4_MUX_WR	—	0x60C	0x60C	—
ioPORT5_MUX_WR	—	0x60D	0x60D	—
ioPORT4_TXAD_WR	—	0x618	0x618	—
ioPORT4_TXST_WR	—	0x619	0x619	—
ioPORT4_RXAD_WR	—	0x61A	0x61A	—
ioPORT5_TXAD_WR	—	0x61C	0x61C	—
ioPORT5_TXST_WR	—	0x61D	0x61D	—
ioPORT5_RXAD_WR	—	0x61E	0x61E	—

**Table 18. Registers (I/O Read) implemented in FPGA**

Register name	I/O Address			Default value
	xDR	JINF	JINJ	
ioBM_DATA_RD	0x000	0x000	0x000	—
ioPORT0_TXAD_RD	0x400	0x400	0x400	—
ioPORT0_TXST_RD	0x401	0x401	0x401	—
ioPORT0_RXAD_RD	0x402	0x402	0x402	0xC000
ioPORT0_RXST_RD	0x403	0x403	0x403	—
ioPORT1_TXAD_RD	0x404	0x404	0x404	—
ioPORT1_TXST_RD	0x405	0x405	0x405	—
ioPORT1_RXAD_RD	0x406	0x406	0x406	0xD000
ioPORT1_RXST_RD	0x407	0x407	0x407	—
ioPORT2_TXAD_RD	0x410	—	0x410	—
ioPORT2_TXST_RD	0x411	—	0x411	—
ioPORT2_RXAD_RD	0x412	—	0x412	0xE000
ioPORT2_RXST_RD	0x413	—	0x413	—
ioPORT3_TXAD_RD	0x414	—	0x414	—
ioPORT3_TXST_RD	0x415	—	0x415	—
ioPORT3_RXAD_RD	0x416	—	0x416	0xF000
ioPORT3_RXST_RD	0x417	—	0x417	—
ioBM_ADWR_RD	0x408	0x408	0x408	—
ioBM_ADDRD_RD	0x409	0x409	0x409	—
ioBM_CRCWR_RD	0x40A	0x40A	0x40A	—
ioBM_CRCRD_RD	0x40B	0x40B	0x40B	—
ioBM_ADDREXT_RD	—	0x40F	0x40F	0x00
ioSEQ_STAT_RD	0x40C	—	—	—
ioSEQ_DATA_RD	0x40D	—	—	—
ioTOF_ADDR_RD	0x40E	—	—	—
ioTOF_DATA_RD	0x40F	—	—	—
ioSSF_RD	—	0x40E	—	—
ioMASKH_RD	—	0x410	—	—
ioMASKL_RD	—	0x411	—	—
ioBUSYH_RD	—	0x412	—	—
ioBUSYL_RD	—	0x413	—	—
ioBERRH_RD	—	0x414	—	—
ioBERRL_RD	—	0x415	—	—
ioDELAY_RD	—	0x416	—	—
ioPORT4_MUX_RD	—	0x40C	0x40C	—
ioPORT5_MUX_RD	—	0x40D	0x40D	—
ioPORT4_TXAD_RD	—	0x418	0x418	—
ioPORT4_TXST_RD	—	0x419	0x419	—
ioPORT5_TXAD_RD	—	0x41C	0x41C	—
ioPORT5_TXST_RD	—	0x41E	0x41E	—

The following internal ADSP-2187L registers are initialized:

**Table 19. Initialization of internal ADSP-2187L registers**

Register name	Default value
IFC	0x00FF
ICNTL	0x0007
IMASK	0x0001
MSTAT	0x0010
PFTYPE	0x4BF0
PFDATA	0x0000
Waitstate	0x7041
SystemControl	0x0407
TPERIOD	0x0064
TCOUNT	0x0064
TSCALE	0x0009

## 21 References

1. ADSP-2100 Family User's Manual. Analog Devices, Inc. 1995;
2. C.H.Lin, AMSWire, AMS-II DAQ Link Protocol, v3.1 4 June 2003;
3. AMS-02 Command and Data Handling. Interface Control Document, Revision 05m. December 10, 2004;
4. D.Haas, C.Zurbach, P.Azzarello. Tracker data reduction;
5. A.Sabellek, Data Processing in UDR2;
6. G.Martinez, RICH data reduction;
7. S.DiFalco, ECAL data reduction;
8. A.Kounine, Data processing in SDR2;
9. C.H.Lin, Trigger Logic design Specification;
10. A.Kounine Data Processing in JLV1;
11. R.J.Lu ADSP Executable file format conversion.