Electrical Ground Support Equipment for the "Obstanovka" Project

SGF Kft.

Prepared by:
Sándor Szalai
János Sulyán
Kálmán Balajthy
Bálint Sódor

Linst of changes:

1. HK decoding:
   PWCconfig.ini th [HKDefinitionFiles] section changed. Description files can be added for a packet and sid pair.
   [experiment ID].[sid] = description file path
2. Graph browsing function:
   A saved waveform can be evaluated in offline mode. It is possible to play back and scroll manually a waveform from a file.

30.03.2007

3. HK decoding:
   Linear interpolation on HK data.

20.05.2009

4. EGSE completion
   Blockscheme of Serial and BITS Telemetry Test Card is added

15.05.2010

5. User interface modification
   DACU1 and DACU2 temperature measuring
   CWZ labels changed to CWD
   Minor modification on preloaded command table

08.12.2011

6. User interface modification
   DACU1 and DACU2 temperature sensor values on dedicated window
   Script file generation bug repaired

22.04.2013

7. Data block management in load data file
8. DACU1 data presentation bug repaired

03.05.2013

9. New functionality in saved data presentation
   Dynamic byte order management based on synchron pattern validation

Content

List of figures

Abbreviations

ADC	Analogue Digital Converter
AGND	Analogue Ground
AMR	Amateur Radio
AMSTM	Analogue Slow Telemetry
APM	Advanced Power Management
ARC	Amateur Radio Channel
AT	Advanced Technology (personal computer)
ATX	Specification for PC motherboard architectures
BIOS	Basic Input Output System
BITS	Bit Serial System
BSC	Binary Synchronous Communication protocol
BSTM	Block of Storage of Telemetry Information Unit
CCSDS	Consultative Committee for Space Data Systems
COM	Computer output (parallel port)
CORES	Correlating Electron Spectrograph (10eV – 10KeV)
CMOS	Complementary Metal Oxide Semiconductor
CPU	Central Processor Unit
CRC	Cyclic Redundancy Code
CWD1, CWD2	Integrated Units of Experiments
CWZ-WP	Combined Wave Sensor
DAC	Digital Analogue Converter
DACU1, DACU2	Data Acquisition and Control Unit
DC	Direct Current
DFM1, DFM2	Flux gate magnetometer
DOS	Disk Operating System
DMA	Direct memory access
DP	Spacecraft potential monitor
EGSE	Electrical Ground Support Equipment
EM	Engineering Model
ETX	End of Text
EPP	Enhanced Parallel Port
FDD	Floppy Disk Drive
FIFO	First In First Out memory organisation
FM	Flight Model
FPU	Floating Point Unit
GB	Gigabyte
GMT	Greenwich Mean Time
Gnd	Ground
GUI	Graphical User Interface
h	Hexadecimal number
HDD	Hard Disk Device
HK	House Keeping
HSO	Hungarian Space Office
Hub	Hub is a place of convergence where data arrives from one or more directions and is forwarded out in one or more other directions

HW	Hardware
IEEE	Institute of Electrical and Electronics Engineers
IF	Interface
IKI	Institute of Space Research (Russian abbreviation)
ISA	Industrial Standard Architecture
ISS	International Space Station
ICS	Internal Control System
ID	Identification
IDE	Integrated Device Electronic (ISA bus developed for IBM PC)
IP	Internet Protocol
I/O	Input / Output
KFKI RMKI	KFKI Research Institute for Particle and Nuclear Physics
LCD	Liquid Crystal Display
LED	Light Emitted Diode
LP	Langmuir probe
MMU	Memory Management Unit
MMX	Multimedia extension
NAK	Negative Acknowledge
OECS	Onboard Electronics Control System
OMTC	Onboard Monitoring Telemetry Channel
OMTS	Onboard Monitoring Telemetry System
PC	Personal Computer
PCI	Peripheral Component Interconnect local bus
PID	Process ID
PWC	Plasma Wave Complex
QM	Qualification Model
RD	Receive Data
RFA	Radio-Frequency Monitor
RS 232/422	Recommended Standard 232/422
RxTx	Receive/Transmit
SAS3	Signal Analyzer and Sample
Sc	Scientific
SID	Structure Identification
SGF	Space and Ground Facilities Ltd
SOH	Start of Header
SPP	Scalable Parallel Processing
STR	Strobe
STX	Start of Text
SVGA	Super Video Graphics Array
SW	Software
TBC	To Be Clarify
TBD	To Be Defined
TC	Telecommand
TD	Transmit Data
TM	Telemetry
TPC/IP	Transmission Control Protocol/Internet Protocol
TTL	Transistor-Transistor Logic

UART	Universal Asynchronous Receiver Transmitter
UIF	User Interface
USB	Universal Serial Bus
VGA	Video Graphics Adapter
VSA	Virtual System Architecture

1. General introduction of Electrical Ground Support Equipment

Concept of Electrical Ground Support Equipment (EGSE) is the name given to the tools required for electrical testing of flight systems. The full checkout of the "Obstanovka" requires several functional units, power supply units and communication channel simulators (onboard Ethernet network, amateur radio channel, bit serial data acquisition system and the so called analog monitoring system). In case of EGSE functions for BSTM and DACU units required the data flow simulators of sensor units. The simulators have to represent the real hardware interfaces. Generally the EGSE of any onboard data acquisition system has four interfaces (see Fig. 1.):

1. User interface to monitor and control the system (display and keyboard);
2. Instrument (space craft) interface, realized on dedicated hardware elements;
3. Data flow source (data simulators of sensors, most cases dummy data flow is satisfactory);
4. Network interface to distribute and archiving the TM data flow (Ethernet).

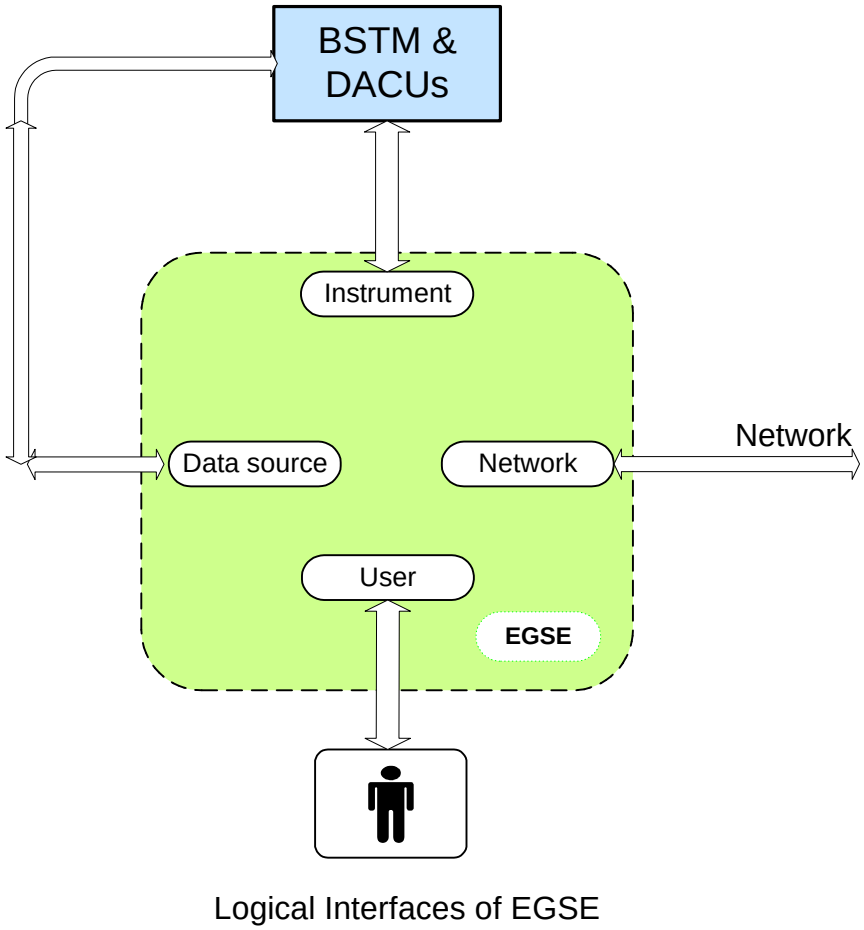


Figure 1 General architecture of EGSE

The PWC-EGSE system simulates the data traffic of the experiments and ISS onboard equipment connected to the PWC computers BSTM, DACU1 and DACU2. The EGSE system consists of an embedded PC104 computer producing the data traffic in real-time, and a connected user interface computer (UIF). This commercially available computer displays the data sent to the ISS onboard system, and enables to switch the power supplies and to send commands and parameters to the experiments on user interaction. The EGSE for Obstanovka (and for its data acquisition and control computers) system consists of two main units: a commercially available computer PC, with Ethernet interface, and a stand-alone box, which contains an ISS signals simulator part (OMTC signals) and simulators of sensor units.

The standalone box realizes a low level simulation of signals connecting to the BSTM and DACUs units. This low level signal simulator box contains a hard disk drive (HDD) to make offline telemetry data read out procedure and gives possibilities to prepare measuring sequences to delivery onboard. The PC software code to be implemented enables the EGSE to process and analyze housekeeping and science data both in real time and from archives in off line mode. The delivered configuration has adequate storage capability for temporary data storage and it will not support permanent data storage. The possible sensor stimulators are not part of the EGSE, they are provided by the experimenter teams.

The Onboard Monitoring Telemetry Interface (OMTC) unit has four different type data acquisition channels:

1. “Analogue housekeeping” data monitoring system simulator;
2. Bit serial digital interface;
3. Amateur radio interface channel;
4. ISS Ethernet network

Data stream acquired by the instrument interface unit is transferred to PC via Ethernet communication channel. The sensor simulators send out adequate signals for BSTM and DACUs. The OMTC simulator and the sensor simulators are built in a common box. The functional units of this stand-alone box are shown on Fig 2. An embedded processor controls both simulators. The processor unit is built on an Intel type microprocessor on which a real time multitasking Linux based operating system runs. The communication between the embedded processor and the PC is going on Ethernet connection using TCP/IP protocol.

The standalone box can be used also as the instrument interface of the Obstanovka system, excluding the sensor simulator part. The “user interface” which realized on the PC by software can be served the function of BSTM & DACUs EGSE and the function of Obstanovka EGSE. On the PC should run Windows 2000 or XP, the dedicated “PWCegse.exe” program gives the user interface. It is a graphical interface to control the system activity and to visualise the telemetry data flow. The software has developed in C language, with the National Instrument firm software, in the integrated development environment (LabWindows/CVI). The configuration of Obstanovka EGSE is shown on Fig. 3. and the configuration to test the BSTM & DACUs is shown on Fig. 4.

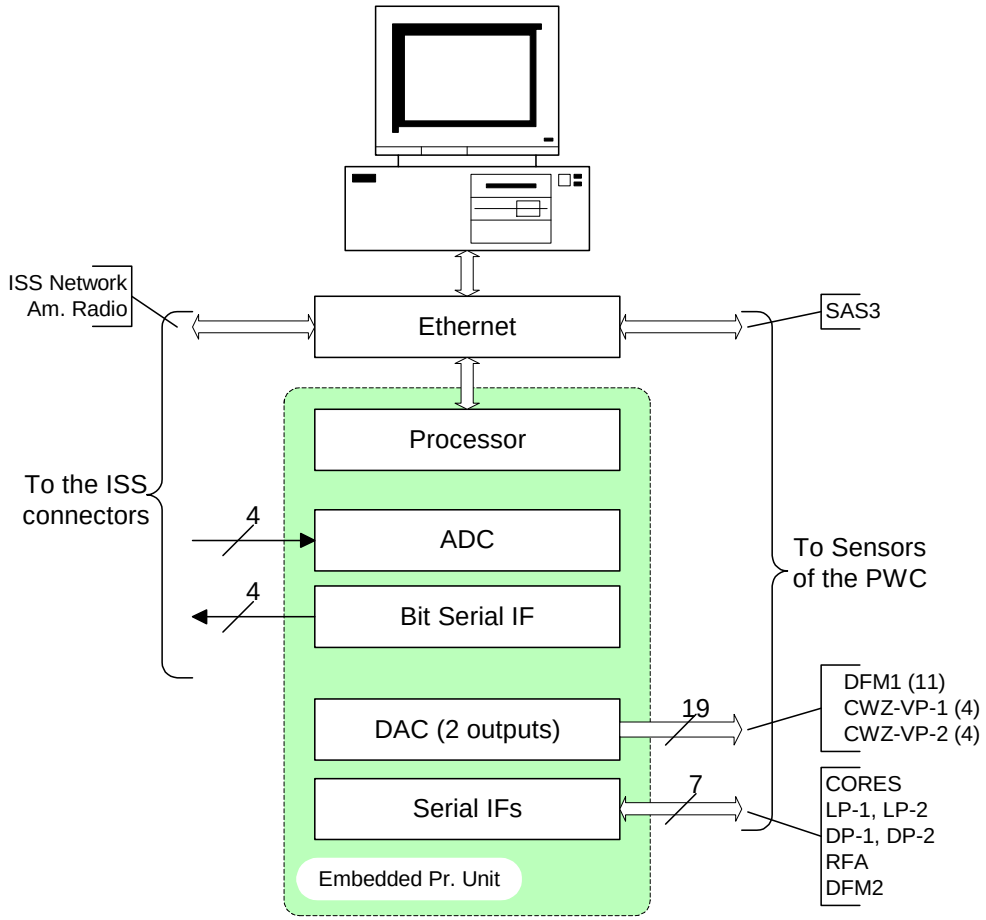


Figure 2 The functional units of the standalone box

2. The EGSE for Obstanovka

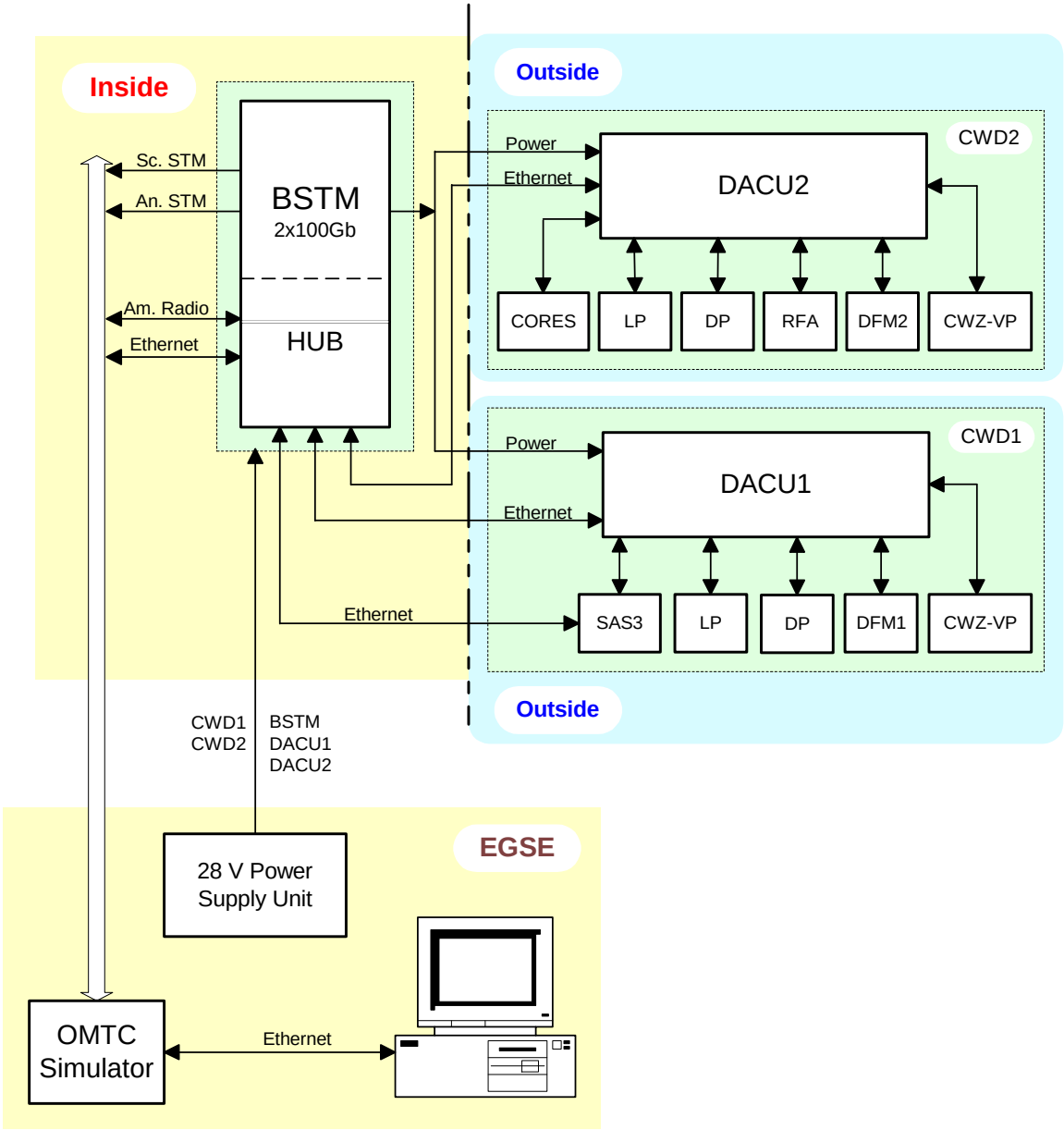


Figure 3 Block diagram of Obstanovka and EGSE configuration

Since Linux is installed on BSTM, DACU1 and DACU2, working under Linux on EGSE too offers lot of advantages, like monitoring active tasks on BSTM, observation Ethernet data traffic and file allocation on every HDD.

Scientific data collected on removable HDD drives are investigated and evaluated by EGSE PC after plugging one 100 GB HDD in HDD Rack of EGSE PC.

EGSE standalone box operating temperature is from +5^{o} C to +35^{o} C. The relative humidity may not more than 80% at 35 C. Nominal input voltage of EGSE is 230V +/- 10, 50 Hz. EGSE will be protected and not to broke down if any failure occurs in EM, QM or FM. Network cable length of EGSE is maximum 5 m. Specifications of input and output circuits of EGSE meet the requirements of onboard electronics. Outputs and inputs of EGSE contain short circuit and over voltage protection.

Set of EGSE contains:

- EGSE
- Complete cables,
- Electronic diagrams,
- Certificate,
- Packaging materials,

Construction of EGSE makes required maintenance possible.

Electric strength between EGSE body and any electronic circuit must be:

- Not less then 20 MΩ at 25^{o} C +/- 10 C ambient temperature and a relative humidity less than 80%
- Not less then 1 MΩ at 25^{o} C +/- 10 C ambient temperature and a relative humidity less than 98%
- Not less then 5 MΩ at 40^{o} C +/- 10 C ambient temperature and a relative humidity less than 80%

Since the engineering model is identical with flight model it is available for requirements of technological tests. The engineering models of BSTM, DACU1 & DACU2 are delivered together with EGSE.

3. The EGSE for BSTM and DACUs

The PWC-EGSE system simulates the data traffic of the experiments and ISS onboard equipment connected to the PWC computers BSTM, DACU1 and DACU2. The EGSE system consists of an embedded PC-104 computer producing the data traffic in real-time, and a connected user interface computer (UIF). This computer displays the data sent to the ISS onboard system, and enables to switch the power supplies and to sent commands and parameters to the experiments on user interaction. The UIF is a commercially available PC having operating system Windows 2000 or higher version.

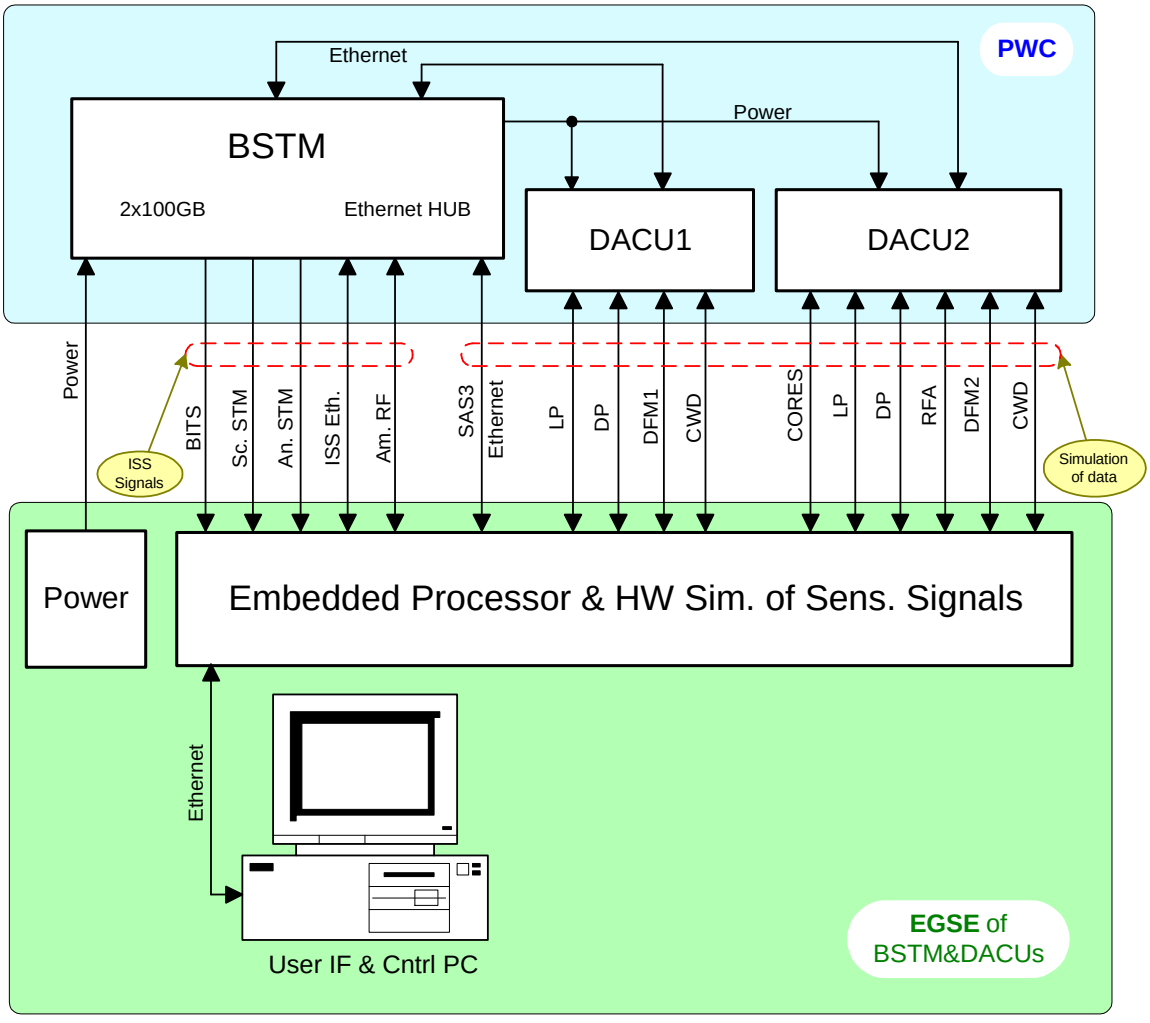


Figure 4 Functional block diagram of the EGSE for BSTM & DACUs without sensors

3.1 Data communication inside the EGSE system

The TCP/IP based communication is used between control PC and embedded processor. The embedded computer acts as server, the User Interface (UIF) PC as client. Port numbers are 5193 for receiving data in the UIF PC, and 5194 for sending data to the embedded computer from control UIF PC. The connection will be realized on the effect pressing 3rd button in the toolbar or from the menu (menu> network > Register In Direction / Register Out Direction / Connect Both Direction). Connection will be demolished by the initialization of operator (menu or the toolbar) or in the case of switching off the embedded processor box of EGSE.

The data structure of the BSTM-DACU communication - between the two parts of the EGSE system - is preserved and extended with the leading “task” field. The task (i.e. the origin / destination) is stored in the low nibble of the HK field of the communication structure.

The EGSE uses the following data structure in the IP communication in both directions:

Name	Length	Description	Remark
PID	2	packet ID	((experiments number) x 16) + 0x880C
Seq	2	sequence count	increasing value | 0xc000
Len	2	data length	length of data bytes – 1, max value is 260
Sec	1	seconds	seconds in time stamp: value = 0 –59
Min	1	minuets	value = 0 – 59
Hour	1	hour	value = 0 – 23
Day	1	day	value = 1 – 31
Month	1	month	value = 1 – 12
Year	1	year	value = 0 corresponds to year 1900
Type	4	packet type	always = 0x30000020
HK	1	Housekeeping	task information in 4 bits, when HK + 0x20
SID	1	structure identifier	values are described by the experiments
Data	261	payload	number of used bytes = len + 1

Table 1.

The data structure (packet) is starting with a synchronization pattern (32 bit). It is key element of the data decoding procedure. Each TM decoding function is starting with the searching of the starting 32 bit long synch word in the data flow from the embedded processor data stream. The synchronization bit pattern was selected by the recommendation of the CCSDS. A study was performed, at the instigation of the CCSDS, to determine a pair of Transfer Frame synchronization markers with relatively low false alarm probability, when these patterns were auto-correlated or pair wise cross-correlated. The tests included auto-correlation of each pattern with itself, with its complement, with its reserve (or mirror), and with the reserve complement. As a result of the above-mentioned tests, the following pair of 32-bit synchronization markers was recommended:

SYNC MARKER 1 (hexadecimal) = 1 A C F F C 1 D

SYNC MARKER 2 (hexadecimal) = 3 5 2 E F 8 5 3

The Sync Marker1 (1ACFFC1Dh) was selected for the Obstanovka experiment.

At the telecommands we have been implemented cyclic redundancy code (CRC) for error detecting. CRC error checking uses a complex calculation to generate a number (word) based on the data transmitted. The sending device (TC preparation unit) performs the calculation before transmission and sends its results to the receiving device (BSTM) in the last word of the TC packet. The receiving BSTM repeats the same calculation; the onboard result will be compared with the last word of the TC. If both calculation results are the same, it is assumed that the transmission was error-free, so the TC is correct and can be executed.

The encoding algorithm is shown on Figure 8. In the encoding algorithm shift registers are used. To encode, the storage stages are set to ’one’, gates A and B are enabled, gate C is inhibited, and (n-16) message bits are clocked into the input. They will appear simultaneously at he output. After the bits have been entered, the output of gate A is clamped to ’zero’, gate B is inhibited, gate C is enabled, and the register is clocked a

further 16 counts. During these counts, the required check bits will appear in succession at the output.

The TCs generated by EGSE are having the starting 32 bit long synchronization marker (Marker 1 = 1ACFFC1Dh) and the closing CRC word. The realized CRC algorithm in C language is the following:

```
//The pointer of the buffer is *pBuff
//iBytes = number of the bytes (length) on which calculates the CRC
void CalculateAndSetCRC(unsigned char *pBuff, int iBytes) {
      int i;
      unsigned short w;
      wCyclRedCode = 0xFFFF;
      for (i=0; i < iBytes-1; i++) {                          //The Length is DataBytes-1
               wCyclRedCode = CrC(pBuff[i], wCyclRedCode);    //Packet CRC Calculation
      }
      pBuff[i++] = (unsigned char)(wCyclRedCode & 0x00FF);    //Low bites part of CRC
      w = (wCyclRedCode & 0xFF00) >> 8;                       //High bites part CRC
      pBuff[i] = (unsigned char)(w & 0xFF);
}
```

The BSTM computer sends powering commands related to the experiments to both DACU units, furthermore telecommands to several experiments. The description of each experiment contains detailed information about the structure of its telecommands. The commands are transferred as the data part of a TM structure message.

Exp	Name	SID	HK	***DACU program***	data description
1	DACU1	0	0	psw_w	power switch command
2	DACU2	0	0	psw_w	power switch commands
4	LP1	0	0	lp_w	telecommands
5	LP2	0	0	lp_w	telecommands
6	DP1	0	0	dp_w	telecommands
7	DP2	0	0	dp_w	telecommands
8	RFA	0	0	rfa_w	telecommands
10	DFM2	0	0	dfm2_w	test mod on / off
11	CORES	0	0	cores_w	telecommands

Table 2.

The data part of the command is different by each experiment, and contains all necessary bytes to be forwarded to that experiment. The power switch commands are treated as an additional “experiment” in this concept.

For example the following command switches the experiment CORES power on:

```
ID = 0x88BC       //The value 0xB = 11 in the bit positions 4-10 means CORES.
                  //The other bits are used to check command validity. The (id &
                  //  0xF80F) must be 0x880F !
Seq = 0xC000      //Not important, but any skip can be reported in the HK data
Len = 0           //Means one byte payload
Sec = 0           //The time and day information is meaningless in case
Min = 0           //   of ISS-PWC communication
```

Hour = 0
Day = 0
Month =0
Year = 0
Type = 0x30000020 //used to check command validity.
HK = 20 //unimportant
SID = 0 //meaningless
Data [0] = 8 // is the required value for the power switch unit to switch
// CORES on
Data[1]…data[260] //are meaningless

The powering commands for the power switch unit are as follows:

Experiment	On	Off
LP	0	1
DP	2	3
CWD	4	5
DFM	6	7
CORES	8	9
RFA	10	11

Table 3.

The detailed commands that control the experiments aren't complete yet. They are / will be defined by the experiments separately. The detailed science data structure of the experiments is also to be defined! The requirements, which science data are to be sent in case of the visibility in different channels, are also to be defined!

The User Interface activates in the embedded system different tasks, the related tasks are:

Task	UIF -> embedded
1	set analog output
2	activate burst mode of SAS3
3	activate burst mode of CORES
4	switch power on / off
5	set DFM2 test on / off
6	CORES test command
7	LP / DP command
8	set radio visibility

Table 4.

3.2 The tasks of the embedded computer

To simulate the digital signals of sensors there is a dedicated board, which has analog inputs too, as illustrated in Figure 5. The Figure 6 shows the simulation board of the analog inputs of the sensor units, it has two analog outputs. In case of all sensor simulation they will be parallelized. On the page 20 and 21 is the circuit diagram of the bit serial telemetry system simulator card. The Ethernet network of ISS and the Amateur Radio interface are the parts of the processor board, 82559 LAN Controller.

There are seven experiments connected via serial lines to the DACU units. Six of them are connected via RS422 interface, one via RS232 interface. The Binary Synchronous Communication (BSC) protocol described in other ISS documents is used with the experiments connected via the RS422 interface. The DFM2 experiment has its own special protocol. The experiment's behavior, the accepted commands and parameters are described in separate documents.

The SAS3 experiment sends data though the Ethernet to the BSTM computer, and informs the DACU1 computer when the "burst mode" of data collection is started. The "burst mode" is activated in the DACU2 computer through a special message from the CORES experiment. The embedded computer has some analog output signals to produce input for the CWD and LP1 experiments.

The ISS onboard system has several communication channels:

- The slow telemetry system measures 6 analog signals in the range of 0 – 10 V with a sample rate of 2 Hz.
- The byte serial interface collects 128 bytes width information with a sample period of 47Hz.
- Ethernet networks

There are two Ethernet channels for communication with Earth:

1. Amateur radio channel, only to send data to the Earth.
2. The ISS onboard Ethernet channel enables a bi-directional connection. Experiments data are sent from the BSTM to the ISS in one direction, and commands arrive from the ISS to the DACUs and experiments in the other direction.

These functions (ISS communication channels and data flow simulation of sensors) are built in the embedded processor system. The function (the logical protocol) realized partly by the software running in the embedded processor and by dedicated hardware units. The functional HW units are shown on the Figure 5.

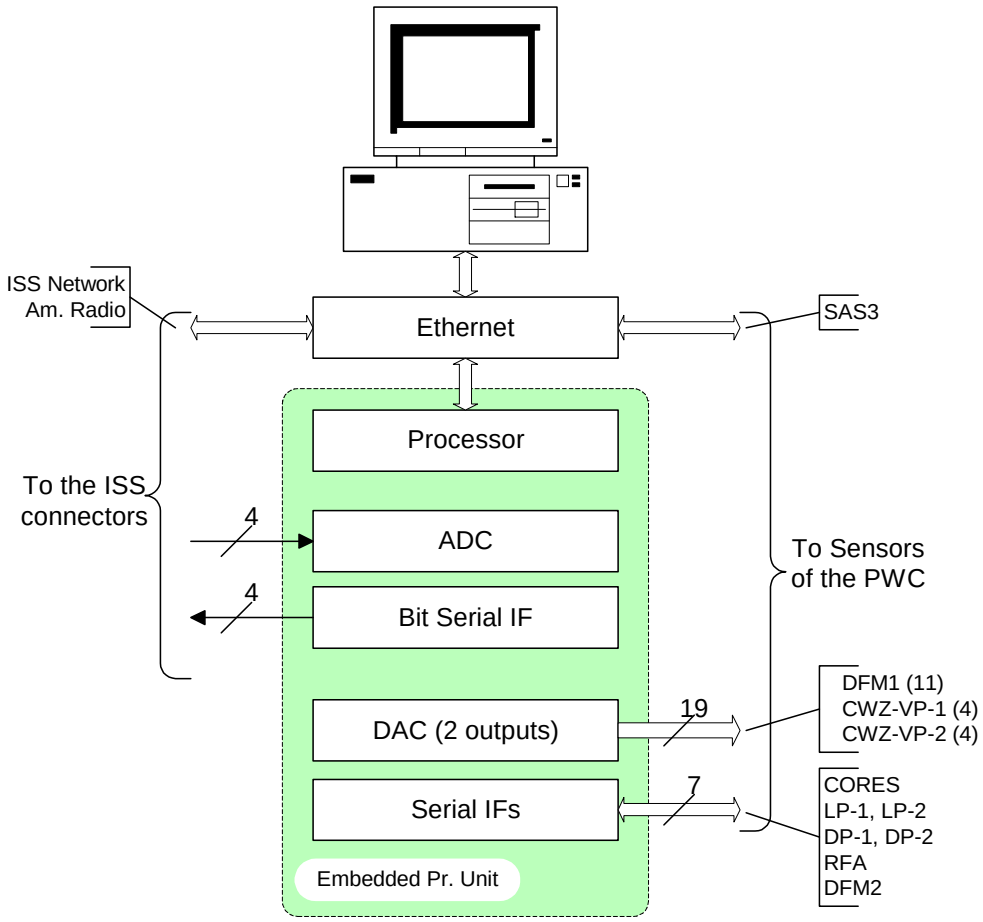


Figure 5 Functional units of the embedded processor box

The order of cards, which realize the low level signal interfaces, in EGSE from the bottom panel to top is:

- PCM-3350 processor card,
- PCM-3618 8-port RS422/485 High-Speed Module; i/o=300h irq5
- PCM-3712 D/A Converter, 2 Channel Analogue Output; i/o=220h, +-10V
- PCM-3718-HG 12 bit DAS Module; i/o=280h
- Slow Telemetry Card (bit serial)

The detailed description of the cards is attached to this document (Appendix 1, 2, 3..)

3.3 The tasks of the user interface computer

The user interface software is running in the Windows 2000/XP multitasking operating system. The PWCegse.exe software is prepared in the National Instruments LabWindows/CVI development environment. The integrated LabWindows/CVI environment features code generation tools and prototyping utilities for fast and easy C code development. It offers a unique, interactive ANSI C approach that delivers access to the full power of C with the ease of use of Visual Basic. Because LabWindows/CVI is a programming environment for developing measurement applications, it includes a large set of run-time libraries for instrument control, data acquisition, analysis, and user interface. LabWindows/CVI also contains many features that make developing measurement applications much easier than developing in traditional C environments. The user interface program runs in multithreading mode.

A multithreaded program is a program in which more than one thread executes the program code at a single time. A single-threaded program has only one thread, referred to as the main thread. The operating system creates the main thread when the operating system begins execute a program. In a multithreaded program, the operating system allows each thread to execute code for a period of time before switching execution to another thread. The period of time during which a particular thread executes is referred to as a time slice. The act of stopping execution of one thread and starting execution of another is referred to as a thread switch. The operating system typically can perform thread switches quickly enough to give the appearance of concurrent execution of more than one thread at a time.

A program that performs data acquisition and displays a user-interface is a good candidate for multithreading. In this type of program, the data acquisition is the time-critical task that might be subject to interference by the user-interface task. In the PWCegse program the TM flow reception is a data acquisition, whish realized trough Ethernet connection. In the PWCegse program the main thread is used to create, display, and run the user interface. Secondary thread is used to perform the time-critical operations such as the data acquisition. LabWindows/CVI provides two high-level mechanisms for running code in secondary threads. These mechanisms are thread pools and asynchronous timers. A thread pool is appropriate for tasks that need to be performed a discrete number of times or tasks that need to be perform in a loop. An asynchronous timer is appropriate for tasks that are to be performing at regular intervals. The thread pool mechanism is used for handling the telemetry channels as Ethernet interface, Radio Amateur telemetry channel and the bit serial channel.

The PWCegse software has a so called panel system. Panel is a separate window having visualization text boxes and control bottoms. The “main” panel has a menu line, a tool bar too. The buttons are to control the PWC activity. The graphical user interface is suitable to control not only the Obstanovka EGSE, but in the BSTM & DACUs EGSE case the simulated sensor activity too. By the PWCegse program different data streams on serial lines and different analogue voltages for DACUs can be simulated.

The user can control the following simulated sensor activity:

- set the analogue outputs
- activate the burst mode message of the SAS3 experiment,
- activate the burst mode message of the CORES experiment,
- switch the power supply of the experiments off / on,
- send test on / off command to the DFM2 experiment,
- send test commands to the CORES experiment,
- send commands and parameters to the LP1, LP2, DP1, DP2 experiments,
- set the visibility mode for the radio channel.

The received data of the EGSE is displayed. Each data source is displayed in separated window. It is organized as a so-called tabulator panel (tabpanel). The data directions are:

- slow telemetry analog measurement,
- bit serial data,
- amateur radio channel, and
- on-board Ethernet.

The control (switch on/off; control of sensors modes: burst, test) of sensors is realized by using buttons. To send different parameters to sensors also should initialize by button and separate panel can be composed (in hex form) the message for the sensors.

3.4 User interface of the control PC

In the default case the PWC EGSE user interface program (PWCegse.exe) is in the C:\Program Files\PWCegse directory, or its icon (Figure 6.) is already on the desktop.

Figure 6 The icon of the PWCegse.exe on the desktop

The program after its starting, it will load the PWCconfig.ini file. The *PWCconfig.ini* file is a readable text file and it is (has to) located in the same project directory (default C:\Program Files\PWCegse). This *PWCconfig.ini* file is a simple interface for storing and accessing hierarchical configuration information using *.ini*-style files or Windows Registry Keys. The *ini* file for the PWCegse has four sections, and each section there are tags. The *.ini*-style files have the following format:

```
[SimIn]
IP = "127.0.0.1"
iPort = 5193

[SimOut]
IP = "127.0.0.1"
iPort = 5194

[CWD]
IP = "192.168.0.100"
iPort = 1000

[DFM]
IP = "192.168.0.110"
iPort = 1001

[RFA]
IP = "192.168.0.120"
iPort = 1002

[DP]
IP = "192.168.0.130"
iPort = 1003

[LP]
IP = "192.168.0.140"
iPort = 1004

[SAS3]
```

```
        IP = "192.168.0.150"
iPort = 1005

[CORES]
IP = "192.168.0.160"
iPort = 1006

[Directories]
TcTm = "\\PWCegse\\current\\pwc\\PWCegse\\src \\tmtc"
AmRadio = "d:\\PWCegse\\current\\pwc\\PWCegse\\src"
XmlFiles = "d:\\PWCegse\\current\\pwc\\PWCegse\\src\\xml"

[HKDefinitionFiles]
Cores.0 = "DefCores.txt"
BSTM.0 = "DefBSTM.txt"
CWD.0 = "DefCWD.txt"
SAS3.0 = "DefSAS3.txt"
SAS3.1 = "DefSAS31.txt"

[TMfilter]
BSTM = 0
DACU1 = 0
DACU2 = 0
SAS3 = 0
LP_1 = 0
LP_2 = 0
DP_1 = 0
DP_2 = 0
RFA = 0
DFM_1 = 0
DFM_2 = 0
CORES = 0
CWD_1 = 0
CWD_2 = 0
HK = 0
SC = 0
HEX = 1

[Cyrillic font]
Name = ""
```

Sections 1 and 2 called “SimIn” and “SimOut” contains the parameters for the TCP/IP connections (TC flow and TM flow) between the BSTM and GUI. Sections 3-9 contains the initial parameters for corresponding data distribution TCP/IP server port. In the above sections. Section 10 (“Directories”) contains the default directories. These values defines the location where the different type of files are placed. Section 11 is the group of the HK Definition files. Each tag defines the file name for decoding of HK packets into readable form. The tag names contains the sensor name and the sid for the

corresponding HK packet separated by a ".". The valid sensor names are: "BSTM", "DACU1", "DACU2", "SAS3", "LP1", "LP2", "DP1", "DP2", "RFA", "DFM1", "DFM2", "CORES", "CWD1", "CWD2" (the naming is not case sensitive). The valid sid number is between 0 and 9. If the definition file, which is declared in the ini file does not exist there will be an error message. The implemented decoding definition for the sensors will be listed in a timed panel (Figure 7.). The *PWCconfig.ini* file can be edited by any character oriented editor. The configuration file will be refreshed with a new content if the new configuration will be saved during the PWCegse program running (Menu Line: 1. Network>Definition of TCP Address>OK; 2. Directory>Save Default Directory; 3. Directory>Save Def. Am.Radio Directory). The HK decoding definition files (section 4) can be edited by an external text editor. The advantages of storing information in this type of standard fashion are:

- Humans can read (and potentially edit) the files and Keys.
- Adding new information to the file does not change its format.

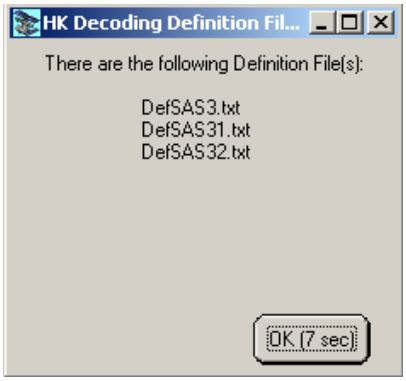


Figure 7 The list of HK decoding definition files

The User Interface is based on the so-called panel (like windows) oriented graphical interface. Panels are a well separated part of the screen and they can be moved and closed. To Closing the main panel (Figure 8.) the control activity will be stopped and for control the PWC the PWCegse.exe program has to run again**.**

There are different areas of the main panel:

1. Menu line
2. Toolbar line
3. Tabpanels
4. Visibility control
5. Sensors' simulation control buttons
6. Sensor control buttons
7. Script Control
8. Quota request and FFT spectrum display buttons
9. TM flow archiving (save to file)
10. Preloaded Binary Commands

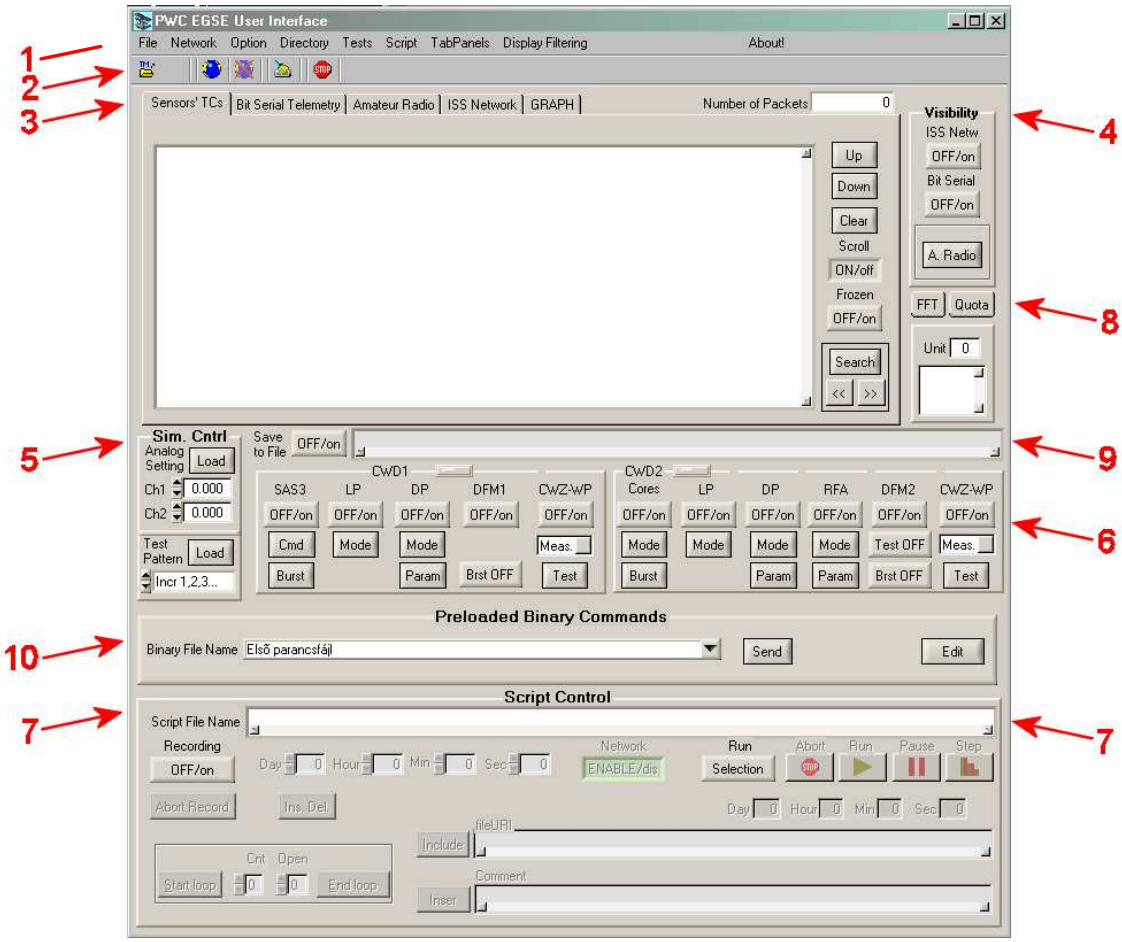


Figure 8 The main panel of PWCegse

3.4.1.Menu line

The menu is a list of options from which a user can make a selection in order to perform a desired action, such as choosing a command or applying a particular format to part of a document. Menu is used as a means of providing the user with an easily learned, easy-to-use alternative to memorizing program commands and their appropriate usage. The menu line has eight menu items. The menu line has the following structure:

File

- Read TM File
- Load HK Decoder File
- Exit

Network

- Register to the Simulator for Telemetry
- Register to the Simulator for Command
- Register TM & Command Direction
- Select Active EGSE for TM
- Install TM Distribution Server

- Data Logging
  - Disconnect Telemetry Direction from Simulator
  - Disconnect Command Direction from Simulator
  - Disconnect TM & Command Direction
  - Uninstall TM Distribution Server
  - Definition of TCP Address

- Option
  - Byte Order for Header
  - Word Order for Header
  - Byte Order for Data
  - Word Order for Data
  - 16 Words in Line
  - 8 Words in Line
  - Log File Size
  - Froze Background Panel
  - Stop by Sections
  - Clear Section
  - Select Cyrillic Font
  - Use Cyrillic Font
  - Display temperatures

- Directory
  - Select Default Directory
  - Save Default Directory
  - Select Def. Am.Radio Directory
  - Save Def. Am.Radio Directory
  - Select Script Directory
  - Save Script Directory

- Script
  - Record > Save
  - Select > Run
  - Generate File
  - Generate HexText File

- TabPanels
  - Sensors' TC
  - Bit Serial Telemetry
  - Amateur Radio
  - ISS Network
  - Chart
    - CWZ1 burst, test
    - CWZ2 test
    - CWZ2 burst
    - CWZ1/2 sid 1
    - DFM1 Ch.1-3
    - DFM1 Ch.4-6

DFM1 Ch.7-11

DFM2
Clear All Box

Display Filtering
Select Filterings' Items
Save Filters' Selection

About

File menu:

In the *File* menu the *Read TM File* item gives possibility to select one earlier stored TM information to study its content in off-line mode. The same possibility is in the Toolbar line (first icon). The *Load HK Decoder File* reloads the definition files of HK packets, this feature is especially useful in preparation phase. The *Exit* item stops the communication on the Ethernet lines and closes the user interface program, this effect can be reached by two other way: 1. press the last icon in the Toolbar line (STOP); 2. close by the Windows usual way selecting the *x* bottom in right upper cornel of the PWCegse panel.

When the PWC control UI application loads a TM file the byte order of the TM data is determined dynamically by validating the synchron pattern at the beginning of every data packet stored in the file.

Network menu:

In the *Network* menu a connection can be initiated (Register to the Simulator for Telemetry, Register to the Simulator for Command or Register TM & Command Direction) or disconnect the PC from the signal level unit, which is the server (Disconnect Telemetry Direction from Simulator, Disconnect Command Direction from Simulator or Disconnect TM & Command Direction). Connect or Disconnect both direction can be done by pressing the appropriate icons in the Toolbar line. The last item (*Definition of TCP Address*) gives possibility to configure the TCP/IP addresses depending of certain conditions of the PWC EGSE system location environment requirements. It can be done in offline mode too. Selective data logging can be performed using the Data Logging option. It means that the incoming data will be stored in different folders based on the source of the data. When user selects the Data Logging option a new window will appear where the data sources for the logging can be selected.

Option menu:

In the *Option* menu the basic option settings can be made. Using the Byte Order for Header, Word Order for Header, Byte Order for Data and Word Order for Data options the word and byte order in the received packets header and data segment can be changed. When the PWC control UI application loads a TM file the byte order of the TM data is determined dynamically by validating the synchron pattern at the beginning of every data packet stored in the file. The 16 Words in Line and 8 Words in Line options are useful for setting the line length in the tab panels' text boxes. During the logging of the received data the program creates a log file and stores the data in it while its size is under a predefined limit. If the size reaches that limit, the system automatically closes it and starts

to store the further data in a new file. The Log File Size option sets this limit. The *Froze Background Panel* option is suitable to freeze the printing to the tab panels witch are not active. If this option is selected than all the data arriving from other sources than the one corresponding to the selected tab panel will not be displayed. The *Display Temperatures* option will display a popup panel with the temperature values of the DACUs.

Directory menu:

In the *Directory* menu the location of the default logging directory and the location of the amateur radios files can be changed.

Script menu:

The Record > Save option in the *Script* menu is appropriate for start the saving of the user interaction is the graphic user interface which can be played using the Select > Run option from the menu.

The Generate HexText File option of the menu generates a command description file using an existiong XML script file. The generated file is a ASCII text file containing the commands to send in hexadecimal format. The file also contains the name of the command and optional comments in ASCII text format.

TabPanels menu:

In the *tab panels* menu sets the visible tab panel. The graph panels are listed under the *Chart* menu. Using the Clear All Box option all the tabpanels can be cleared.

3.4.2.Toolbar line

The Toolbar line has Icons as buttons, when these icons are clicked on with mouse, certain functions of the menu line items are activated. The icons make short cut to the File > Read TM File, to Network > Register TM & Command Direction, to Network > Disconnect TM & Command Direction; and to File > Exit menu items correspondently of the order of icons. The fourth icon activation clears all tabpanel textboxes and the CWD graph. The last icon finishes the running of the PWCegse program itself.

The activity of the Obstanovka experiment is determined by the cyclorama running in the BSTM. The cyclorama is a sequence of commands, which are executed in the given time. This type of sequences can be prepared in the EGSE and they execution can be tested. The sequence file is a readable XML type file, and it can be prepared by any of Editor Software or any of dedicated XML Editors, but it can be prepared by pressing the buttons in the required sequences and inserting appropriate delay time between the actions/commands. Using the EGSE User Interface to prepare the Obstanovka sequence also generates an XML file. The generated file is the so called script file. This file should be converted for the BSTM (excluding synchro pattern, CRC word and control of the sensor simulators) and using the stand alone box of EGSE will be writes by Linux system into the HDD, which will be inserted into BSTM unit. The syntax and the detailed description of the XML script file are detailed in chapter 5.

HK packets of the different sensors can be visualized into readable text form (instead of Hex Dump). The conversion is working on the base of external definition files.

The detailed syntax of the definition file is described in chapter 6. It gives possibility to visualize the parameters in decimal or hexadecimal forms.

Any TM file can visualized in tabpanels with drag and drop technique. You can select the actual file (from a file lists of any windows program: Windows Commander, Total Commander, Windows Explorer) and keeping pressing down the left button of mouse moving (dragging) it into the PWCegse main panel. Release the mouse button and the selected file will be processed and the result will be displayed in the appropriate text box.

3.4.3. Tabpanels

The data traffic of the different hw/sw interfaces is displayed in the Tabpanel window. There are five different overlapping panels. The firs four panels corresponding to the different data sources while the last panel shows the CWDs graph:

1. Sensors' TC,
2. Bit Serial Telemetry,
3. Amateur Radio,
4. ISS Network.
5. Graph

The Sensors' TC panel displays data received by the simulated experiments.
The Bit Serial Telemetry panel shows data transferred through that interface.
The Amateur Radio panel displays the content of one selected file.
The ISS Network panel shows data sent to that interface.
Graph displays the CWD1, CWD2 and DFM1 instruments signals from different channels. The system refreshes a chart only if it is visible. In off-line (disconnected state) the Graph panel gives the ability to process a archive file and browse the waveforms saved in the selected file. This can be done via the 'SELECT' button or simply 'drag and drop' the file to the Graph panel.

Each panel preserves its own control settings. The operator can enable or disable the scrolling and the refreshing of the panel, switching the Scroll Off/On or freezing the panel. A frozen panel is not refreshed; the new data will not display on it. When Scroll is On, any new data will appear in the bottom of the panel, and the last information will be displayed, even if previously the operator scrolled it up. It is possible to scroll over the panel by the Up and Down buttons or dragging the scrollbar on the right side. It is recommended to switch the Scroll to the Off state or freeze the panel before examining the content of the panel. Using the Search facility, a search pattern can be set and searched forward or backward.

The content of the active panel can be cleared pressing the Clear button.

The settings of the Option and Display Filtering menus apply to the four panels simultaneously.

If a saved data file is loaded via either the *Read TM File* in *File* menu or by drag-and-drop on the main panel the *File browse* pop-up panel will appear. The load function

loads only the first data block from the TM file, the following blocks can be loaded by using the step button on the *File browse panel*.

Figure 9 File browse panel

3.4.4. Visibility control

The ISS Network and the Bit Serial Interface can be activated or deactivated separately. When one is active, it accepts data; otherwise the interface is inactive.

Selecting the A.Radio button, the directory containing the amateur radio data files can be selected and displayed. Selecting and opening a file from it, the content of this file will be displayed in the Amateur Radio panel.

3.4.5. Sensors'simulation control

The CWD and DFM1 experiments produce analog signals measured by the DACUs. There are two analog outputs available in the EGSE to simulate analog outputs. First set the required value in the range +/- 10 V, and press the Analog Setting Load button.

The simulated LP, DP, CORES, RFA, SAS experiments produce dummy science data packets. The content of these packets can be selected from the predefined items (all 00, all ff, increasing bytes, etc.), and the selected Test Pattern can be loaded to the experiments simulator.

3.4.6.Sensors' control

A green box indicates the communication of DACU units with the BSTM.

The On/Off button of the sensors indicates with green color the powered state of the sensor, evaluating the slow telemetry analog signals. Pressing the button, the sensor will be switched off or on alternatively.

Some experiments have the possibility to be controlled:

Pressing the Burts button of the SAS3 experiment, the burst data sampling mode of the DACU1 unit will be activated through the SAS3 simulator.

The SAS3 experiment accepts some parameters. To compose them press the Cmd button under the SAS3 label. The desired parameters can be set on the activated panel (see

Fig. 9). Press the Send button to send the parameter setting command to the SAS3. Use the File Selection button to load settings from file.

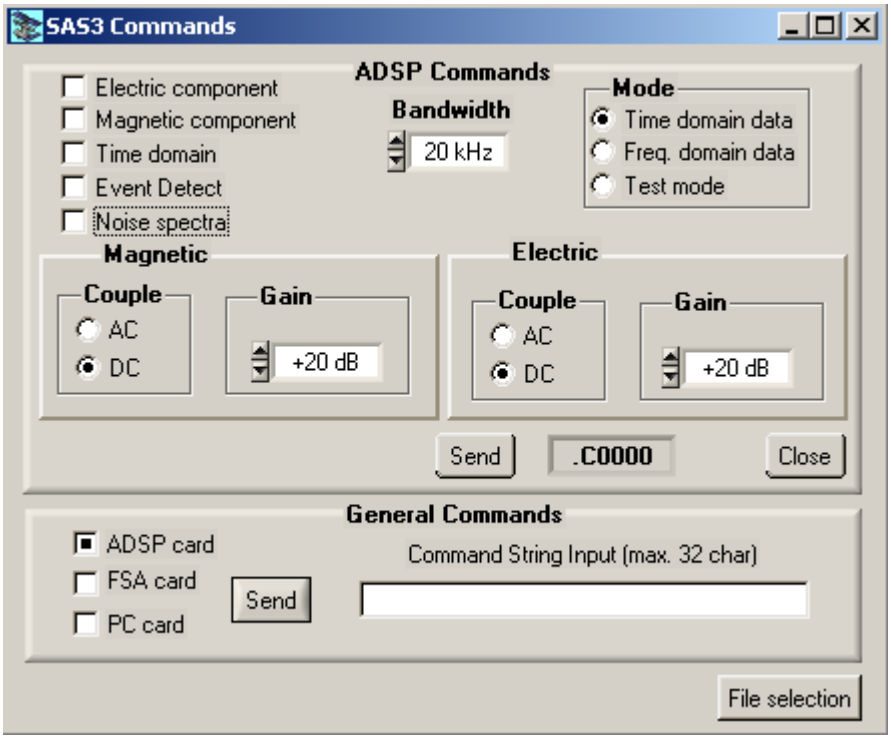


Figure 10 The SAS3 commands

The LP, DP and RFA experiments accept parameters and mode setting commands. The mode commands are to be defined! Pressing the Param button under the sensor's label the Parameter Definition panel appears (see Fig. 10). Type in two digit hexa values in the input field. The parameter line can be saved into a file pressing the Save button. A previously saved parameter can be loaded pressing the Open button. Pressing the Clear button clears the input field. Pressing the OK button, the parameter will be sent.

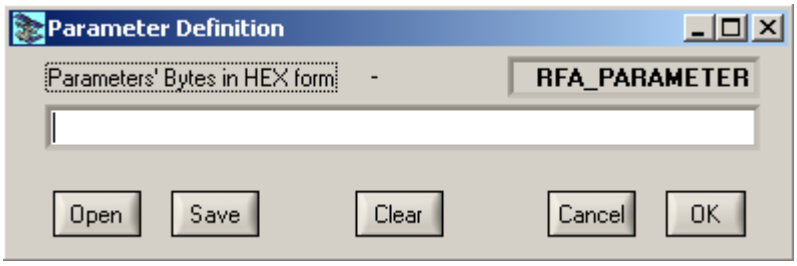
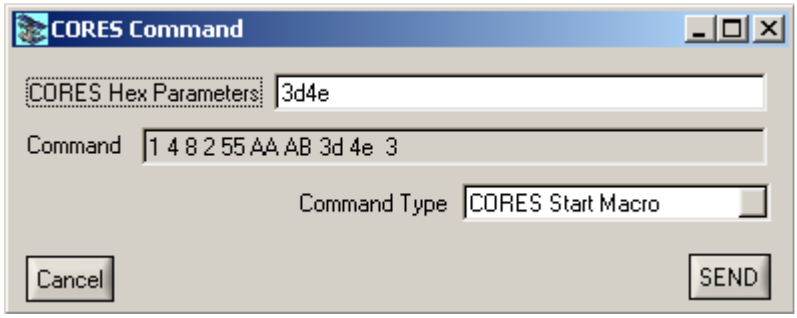


Figure 11 Parameter definition

The CORES experiment simulator generates the burst data sampling mode command for the DACU2 unit by pressing the Burst button under the Cores label.

Pressing the Mode button under the Cores label the Cores Command panel appears (see Fig. 11). Select the desired command type then fill the CORES Hex Parameters field. The send button will be active only if the parameters corresponding to the selected command type are typed. Note that the Command filed indicates the complete command to be sent. The Parameters are always in hex format.



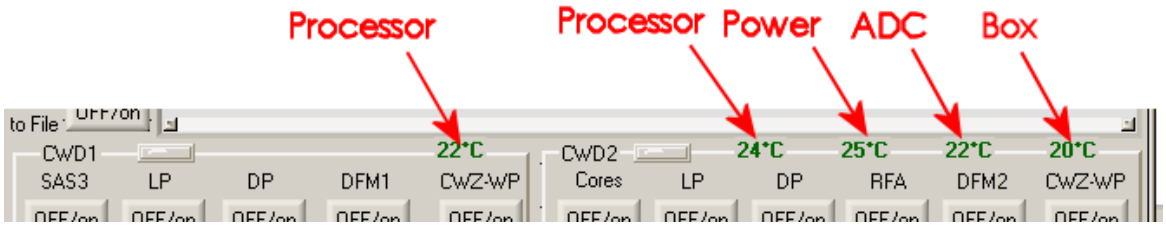

```
<SetCORES_Global_Mode value="ef"/>
<SetCORES_Software parameter="a2" value="3f"/>
<CORES_Start_Macro msadd="54" lsadd="e3"/>
```

Figure 12 CORES Command window and XML example

The test mode of the DFM2 experiment can be switched on / off pressing the Test M button under the DFM2 label.

There is a way to send telecommands prepared previously. Select a sensor under the TC File selection button, and press the button. Select a telecommand file to be downloaded. The file will be segmented, if necessary.

The temperatures of processor of DACU1 indicated on the frame of sensors' OFF/on switches and in DACU2 there are four sensors in different points. Temperatures of the measured point are shown in the following figure:



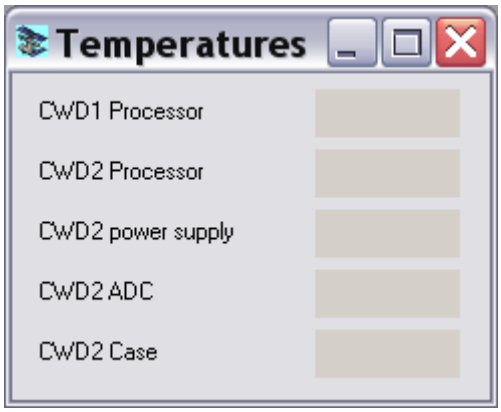


Figure 13 values a.) on main panel; b.) on Temperatures panel

The actual temperatures of the DACUs can be displayed on a dedicated *Temperatures* panel. The Temperatures panel can be opened using the *Display Temperature* option in option menu. The Temperatures panel displays the 5 different temperatures with the location of the temperature measurement inside the DACUs.

3.4.7. Script control

The sensor's control command described in the previous section can be saved in a so-called script file. It is possible later to "execute" this file. The scipt file is a text file, with .xml extension. It can read and edited later with any text editor.

To make a new script file first press the Selection button. First of all decide, whether the commands should be sent to the BSTM or not. Answer No on the "Do you want to control during recording?" question, if you are not connected to the BSTM. After it select the directory and give a new file name to the new script file, or select an existing one, to overwrite it. Fill the header of the file in the next panel (Created by and Comment fields). Press Start saving. The Recording button changes to green ON/off, indicating that the following commands will be saved. Pressing this button will finish the recording and closes the script file.

During recording any sensor control command will be saved. This commands are:

1. Powering commands
2. SAS3 Commands
3. LP, DP, RFA Mode and Param commands
4. CORES Mode command
5. DFM2 Test M

The Burst commands are controlling the simulator, and not the sensors, so they are not saved.

Delays can be inserted between the telecommands: set the delay value and press the Ins.Del. button. Unless delays the script actions will be executed continuously.

There are some advanced features too: Loops, even embedded loops can be defined. First select the Count, then press Start loop. The Opened counter shows the loop depth. Close the actual loop level pressing the End loop button. The next useful feature is the include facility: press the include button and select a script file. During execution the

commands of the included file will be executed from this point. After the last command of the included file the control returns to the next command of the original script file. Of course an included file itself can contain other includes.

To execute an existing script file first press the Run Selection button and select the script file. The script file name appears, the label of the Selection button changes on Selected. Press the Run > button to start the execution. The label color and text changes on green Running. Now the execution can be aborted, paused or suspended to manual step-by step execution by pressing the appropriate button. The execution can be started also by pressing the Step button. During execution it is always possible to change between Run, Step and Pause state.

3.4.8. Quota and FFT buttons

Switching the Quota button On forces the BSTM to send data to the ISS Ethernet and to the Bit serial interfaces without a new predefined quota limitation. This function is useful for test purposes in connection with the TM flow archiving.
FFT button displays the spectrum window and enables the Fast Fourier Transformation for a set of channels. The spectrums are counted for the CWD1 channel 1, channel 2 and channel 3, CWD2 channel 1, DFM1 channel 1, channel 2 and channel 3.

3.4.9. TM flow archiving (save to file)

Press the Save to File button. The automatically generated file name will be displayed. All received packets will be stored from this moment on. To close the file, press the button again. The file will be closed automatically at size 1.4 M byte. If the file can't be opened, you should select (and save) the Default Directory in the Directory menu.

3.4.10. Amateur Radio TM flow reception

On the ISS there will be a separated computer for amateur radio connection using Windows operating system and acting as an FTP file server. The BSTM computer transfers scientific data packages to this computer in every hour using the FTP protocol. For every sensor there will be a separate file with maximum length of 50 kilobyte. The EGSE simulates the amateur radio computer - it acts as an FTP file server. The transferred files can be examined using the user interface, or copied to another computer.

3.4.11.Preloaded Binary Commands

The so called "preloaded binary command files" are command sequence files stored on the BSTM. Each of these files describes a typical command sequence. In case of a typical command sequence a preloaded command file can be executed from the GUI with one TC instead of sending the commands of the typical sequence one by one. The preloaded binary file execution TC contains only the id (1 byte) of the command sequence file. Inside the GUI software there are a lookup table witch links the different ids with a

short description. So the selection of binary files can be done using these short descriptions instead of the plain id.

The “Binary File Name” list contains the short descriptions of the binary files on the BSTM. With the “Send” button the execution of the selected file can be started. Using the “Edit” button the lookup table can be edited. It is the operators task to maintain a valid lookup table.

Sending ‘Start Preloaded Binary File’ commands can be saved into command sequence file and can be played back as well. When running an .xml script file with a command sequence containing a ‘Start Preloaded Binary File’ command, it is recommended that the EGSE GUI program contains the same preloaded file table as the one which generated the script file. To aid this recommendation the GUI software uses a *BFconfig.ini* file which contains the full preloaded binary file table. For convenience this file can be exported end imported between different EGSE GUI software using the *export* and *import* buttons on the ‘Edit Binary Files’ window.

Installing FTP Server on EGSE

For the FTP server in EGSE, a computer with a Microsoft XP operating system, a GNU licensed software called FileZilla is applied. FileZilla is accessible on the Internet, at the address http://sourceforge.net/projects/filezilla/. At this site, there could also be found the binaries for Microsoft’s operating systems Win95, Win98, Windows NT4, 2000 and XP, the documentation FileZilla, and the FileZilla’s sources codes.

On EGSE, the FileZilla version 0.9.11.0 (built on 2005-11-13 15:10) is installed and configured, below the “C:\Program Files\FileZillaFTP” directory. The installation itself is quite simple, the process is a menu driven. The downloaded FileZilla_Server-0.9.11.exe should only be started keeping its default settings.

The configuration file of the FileZilla is an XML formatted file. It is named “FileZilla Server.xml”. This XML file is mostly self-explanatory. It is located under the installation directory of the FTP server, namely “C:\Program Files\FileZillaFTP”.

The server configuration could be set up editing this XML file or using the graphic user interface called “FileZilla Server Interface.exe”. For transferring data from BSTM to EGSE, the Anonymous FTP user was created with password. The working directory for this user is the directory called C:\FileZillaFtp\Documents\Anonymous. For the Anonymous user, the full access right was granted on this directory.

When the EGSE is connected to an open “network”, firewall software has to be set up and switch on. If this is the case, an FTP connection has to be permitted for BSTM.

Installing FTP client on BSTM

On the BSTM, the FTP client was installed from the software packages of SuSE 8.1. The ftp executable is located below the directory /usr/bin. The configuration file for the ftp connections is called .netrc. This contains the machine name or its IP address, the user id and the user’s password. The file is located in the home directory of the user executing the ftp command.

"FileZilla Server.xml"

```
- <FileZillaServer>
  - <Settings>
      <Item name="Serverport" type="numeric">21</Item>
      <Item name="Number of Threads" type="numeric">2</Item>
      <Item name="Maximum user count" type="numeric">0</Item>
      <Item name="Timeout" type="numeric">120</Item>
      <Item name="No Transfer Timeout" type="numeric">120</Item>
      <Item name="Allow Incoming FXP" type="numeric">0</Item>
      <Item name="Allow outgoing FXP" type="numeric">0</Item>
      <Item name="No Strict In FXP" type="numeric">0</Item>
      <Item name="No Strict Out FXP" type="numeric">0</Item>
      <Item name="Login Timeout" type="numeric">60</Item>
      <Item name="Show Pass in Log" type="numeric">0</Item>
      <Item name="Custom PASV Enable" type="numeric">0</Item>
      <Item name="Custom PASV IP" type="string" />
      <Item name="Custom PASV min port" type="numeric">0</Item>
      <Item name="Custom PASV max port" type="numeric">0</Item>
      <Item name="Initial Welcome Message" type="string">%v written by Tim Kosse (Tim.Kosse@gmx.de) Please visit http://sourceforge.net/projects/filezilla/</Item>
      <Item name="Admin IP Bindings" type="string" />
      <Item name="Admin IP Addresses" type="string" />
      <Item name="Enable logging" type="numeric">0</Item>
      <Item name="Logsize limit" type="numeric">0</Item>
      <Item name="Logfile type" type="numeric">0</Item>
      <Item name="Logfile delete time" type="numeric">0</Item>
      <Item name="Use GSS Support" type="numeric">0</Item>
      <Item name="GSS Prompt for Password" type="numeric">1</Item>
      <Item name="Download Speedlimit Type" type="numeric">0</Item>
      <Item name="Upload Speedlimit Type" type="numeric">0</Item>
      <Item name="Download Speedlimit" type="numeric">10</Item>
      <Item name="Upload Speedlimit" type="numeric">10</Item>
      <Item name="Buffer Size" type="numeric">4096</Item>
      <Item name="Admin port" type="numeric">14147</Item>
      <Item name="Serverports" type="string">21</Item>
      <Item name="Custom PASV IP type" type="numeric">0</Item>
      <Item name="Custom PASV IP server" type="string">http://filezilla.sourceforge.net/misc/ip.php
      </Item>
```

```
        <Item name="Use custom PASV ports" type="numeric">0</Item>
      <Item name="Mode Z Use" type="numeric">0</Item>
      <Item name="Mode Z min level" type="numeric">1</Item>
      <Item name="Mode Z max level" type="numeric">9</Item>
      <Item name="Mode Z allow local" type="numeric">0</Item>
      <Item name="Mode Z disallowed IPs" type="string" />
      <Item name="IP Bindings" type="string">*</Item>
      <Item name="IP Filter Allowed" type="string" />
      <Item name="IP Filter Disallowed" type="string" />
        <Item name="Hide Welcome Message" type="numeric">0</Item>
      <Item name="Enable SSL" type="numeric">0</Item>
      <Item name="Allow explicit SSL" type="numeric">1</Item>
      <Item name="SSL Key file" type="string" />
      <Item name="SSL Certificate file" type="string" />
      <Item name="Implicit SSL ports" type="string">990</Item>
      <Item name="Force explicit SSL" type="numeric">0</Item>
        <Item name="Network Buffer Size" type="numeric">65536</Item>
    - <SpeedLimits>
        <Download />
        <Upload />
      </SpeedLimits>
    </Settings>
    <Groups />
  - <Users>
    - <User Name="anonymous">
        <Option Name="Pass" />
        <Option Name="Group" />
        <Option Name="Bypass server userlimit">0</Option>
        <Option Name="User Limit">0</Option>
        <Option Name="IP Limit">0</Option>
        <Option Name="Enabled">1</Option>
        <Option Name="Comments" />
      - <IpFilter>
          <Disallowed />
          <Allowed />
        </IpFilter>
      - <Permissions>
        - <Permission Dir="C:\FileZillaFtp\Documents\Anonymous">
            <Option Name="FileRead">1</Option>
            <Option Name="FileWrite">1</Option>
            <Option Name="FileDelete">1</Option>
            <Option Name="FileAppend">0</Option>
            <Option Name="DirCreate">1</Option>
            <Option Name="DirDelete">1</Option>
            <Option Name="DirList">1</Option>
            <Option Name="DirSubdirs">1</Option>
            <Option Name="IsHome">1</Option>
```

```
        <Option Name="AutoCreate">1</Option>
          </Permission>
        </Permissions>
      -       <SpeedLimits      DlType="0"      DlLimit="10"
        ServerDlLimitBypass="0"     UlType="0"     UlLimit="10"
        ServerUlLimitBypass="0">
          <Download />
          <Upload />
        </SpeedLimits>
      </User>
    </Users>
  </FileZillaServer>
```

“.netrc”

```
machine amr
login anonymous
password anonymous
```

3.5 Program modules of embedded processor

A LINUX based dedicated real-time multitasking operating system is running, which has the following application tasks. The operating system is resident in the flash memory of the embedded processor board.

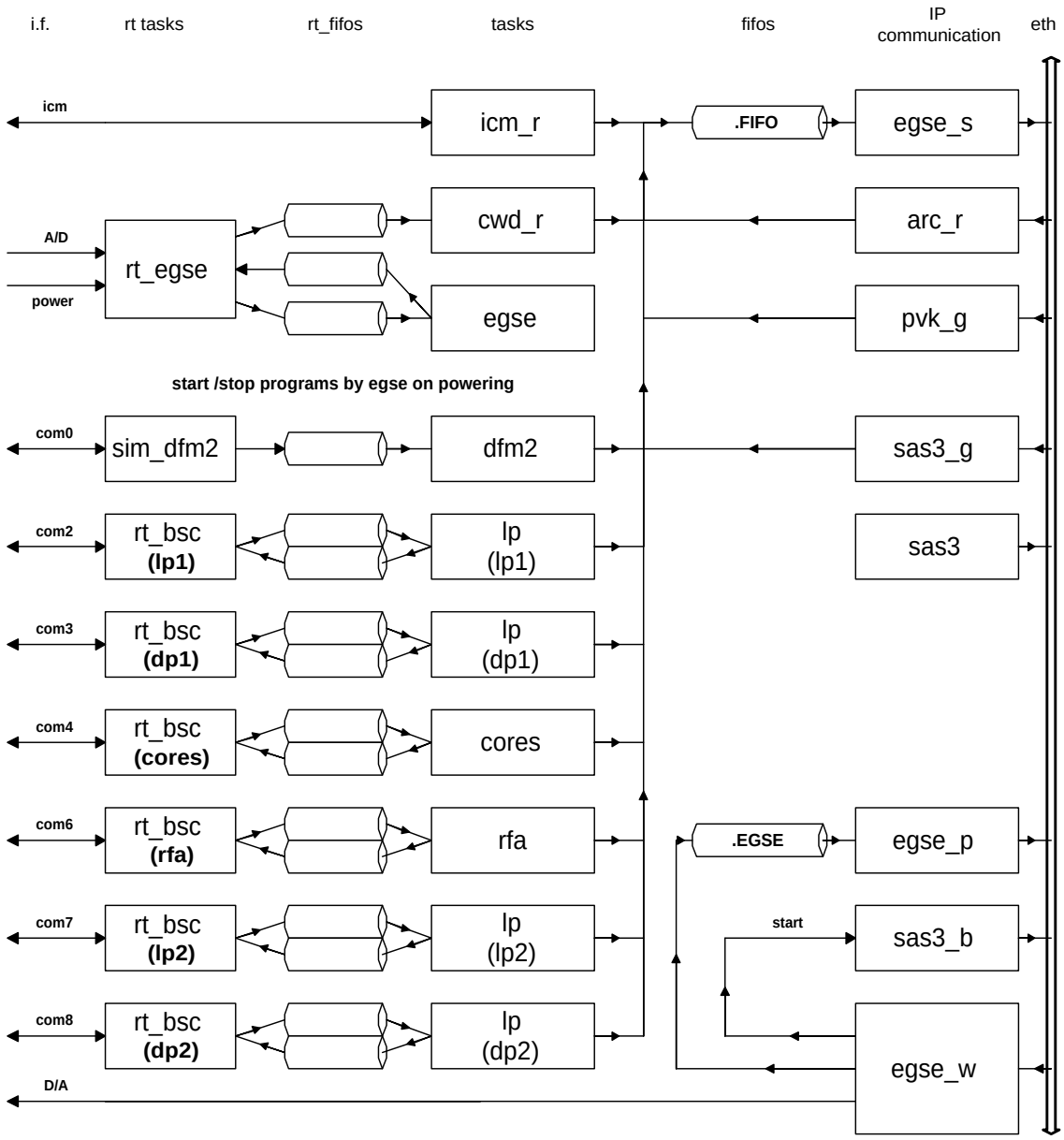


Figure 14 Structure of the embedded software

rt_egse

This program measures the bits analog signals with one Hz sampling rate. It senses the change of the powering signals of the experiment.

egse

This program starts or stops the experiments depending on the state of the powering signals. The following experiments are simulated: LP1, LP2, DP1, DP2, CORES, RFA, DFM2, SAS3.

cwd_r

This program sends the measured analog values (bits) to the fifo .FIFO.

read_send

This program reads data from the fifo .FIFO, and forwards them through the Ethernet to the UIF computer.

egse_w

This program accepts commands from the UIF computer. Depending on the command it sets analog outputs, starts the sas3 burst mode or forwards the command to the fifo .EGSE.

egse_p

This program reads data from the fifo .EGSE and forwards them through the Ethernet to the BSTM computer.

lp

Produces data trafic of the lp, dp, rfa experiments.

cores

Produces data trafic of the cores experiment.

dfm2

Produces data trafic of the dfm2 experiment.

sas3

Produces data trafic of the SAS3 experiment.

sas3_b

Generates a burst mode message from the sas3 experiment.

icm_r

Accepts data from the low telemetry system and forwards them to the FIFO.

4. Functional block diagrams of cards of the embedded system

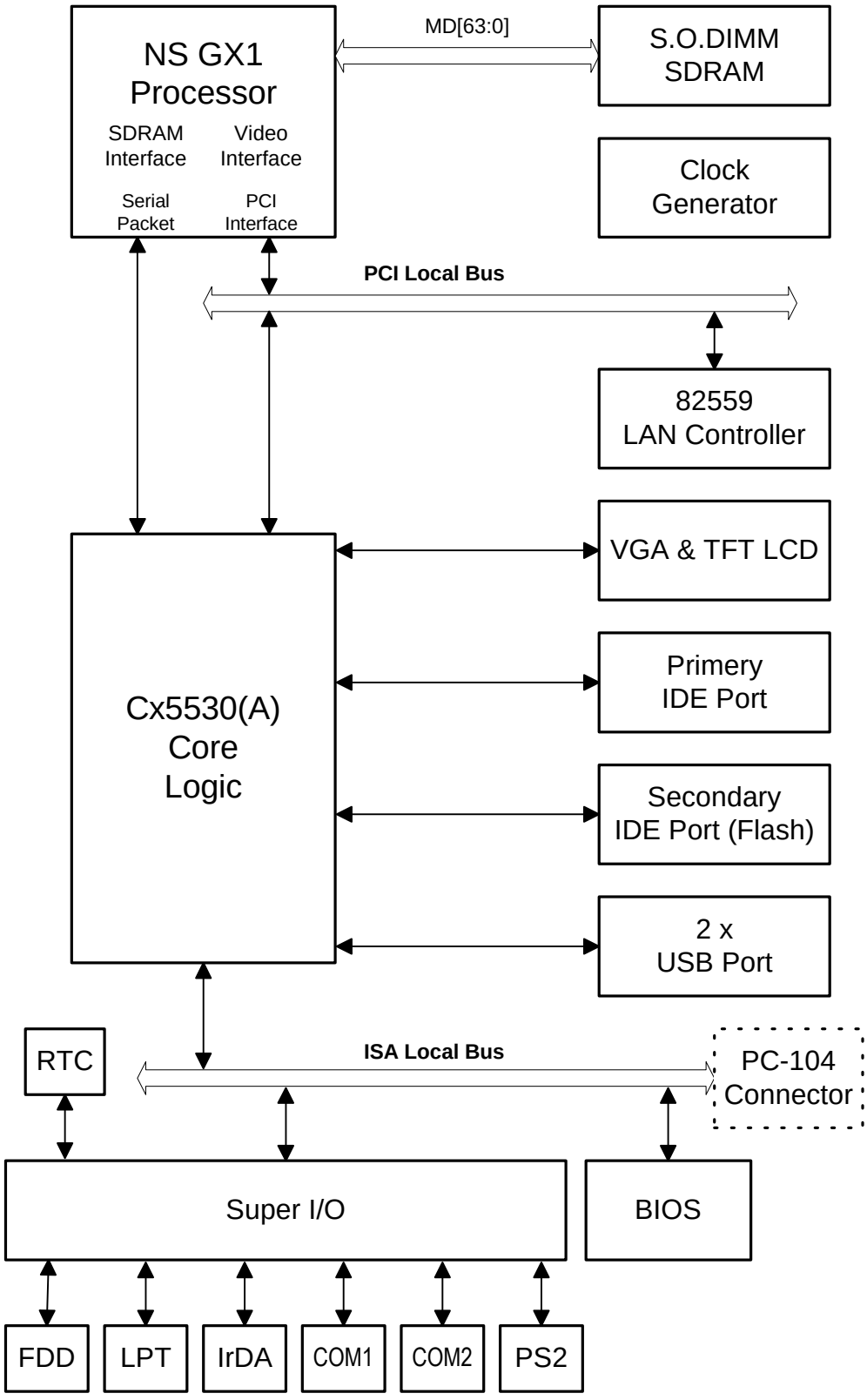


Figure 15 Functional Blocksheme of Processor Card

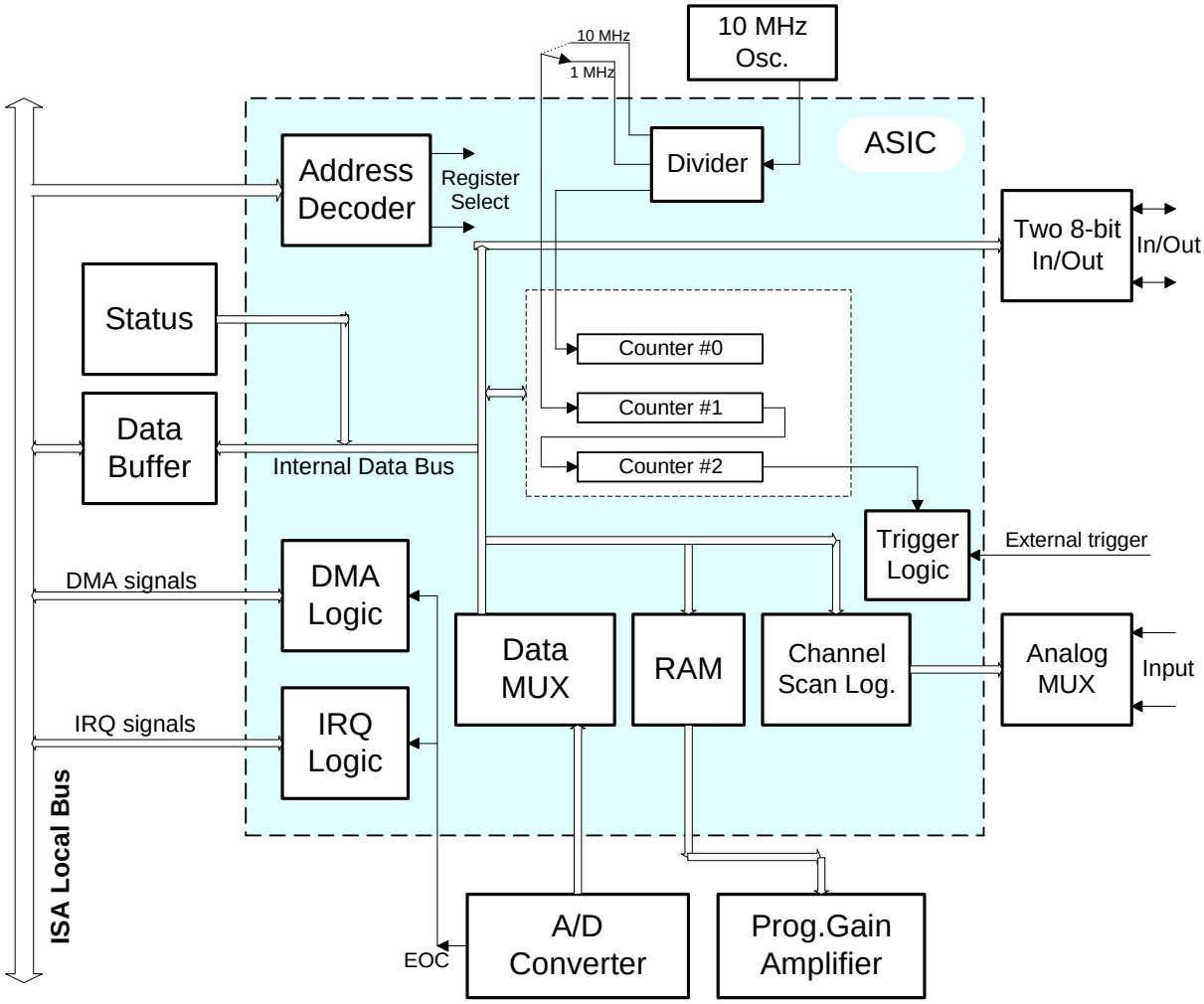


Figure 16 Block diagram of the12-bit DAS Module with Programmable Gain

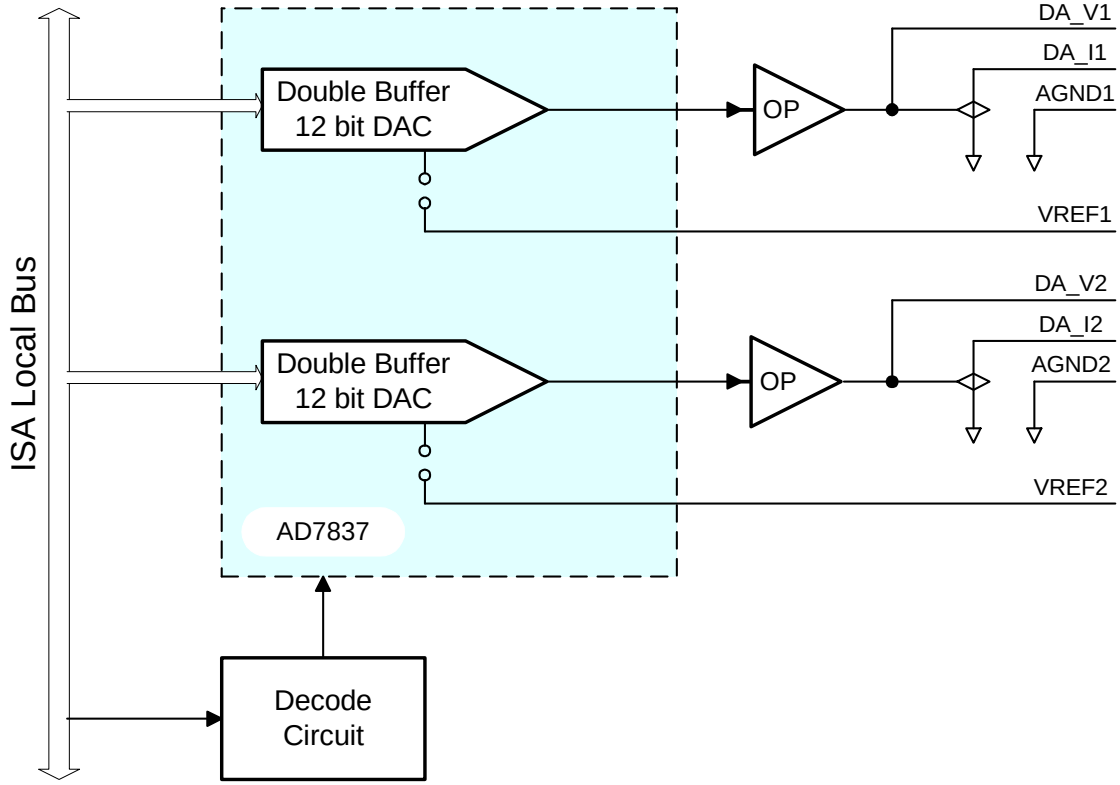


Figure 17 Functional Blockcheme of the two channel analog output card

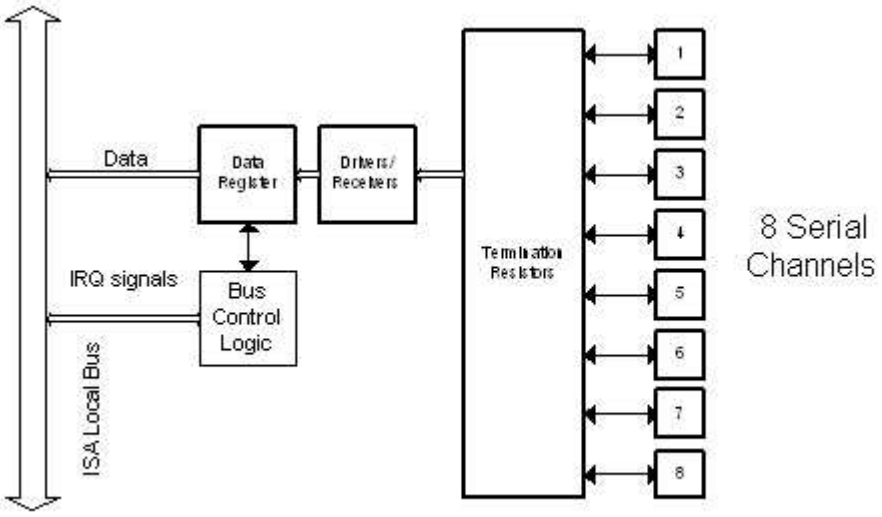


Figure 18 Functional Blockcheme of the Eight Channel Serial card

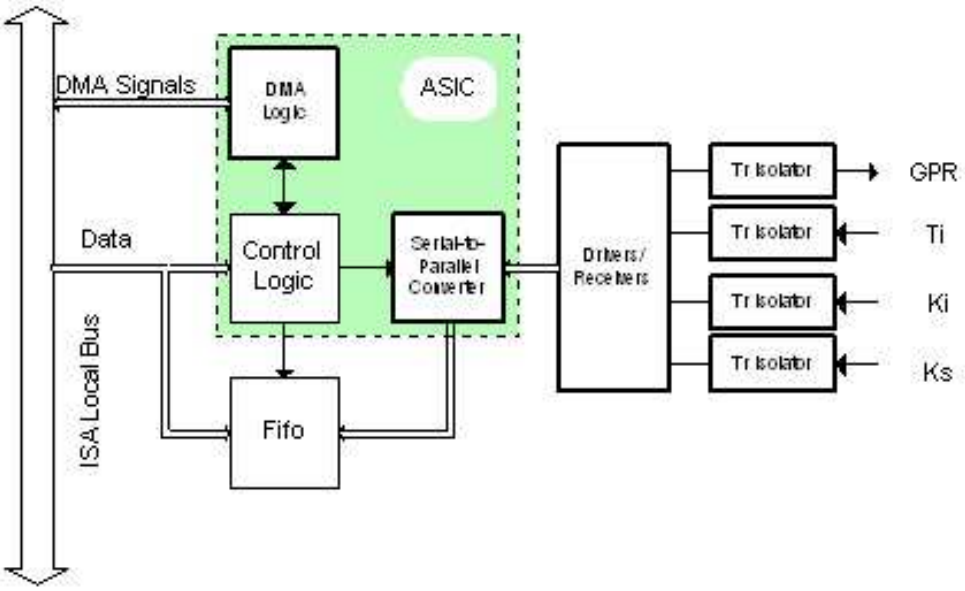


Figure 19 Functional Blockcheme Telemetry Test card of BITS

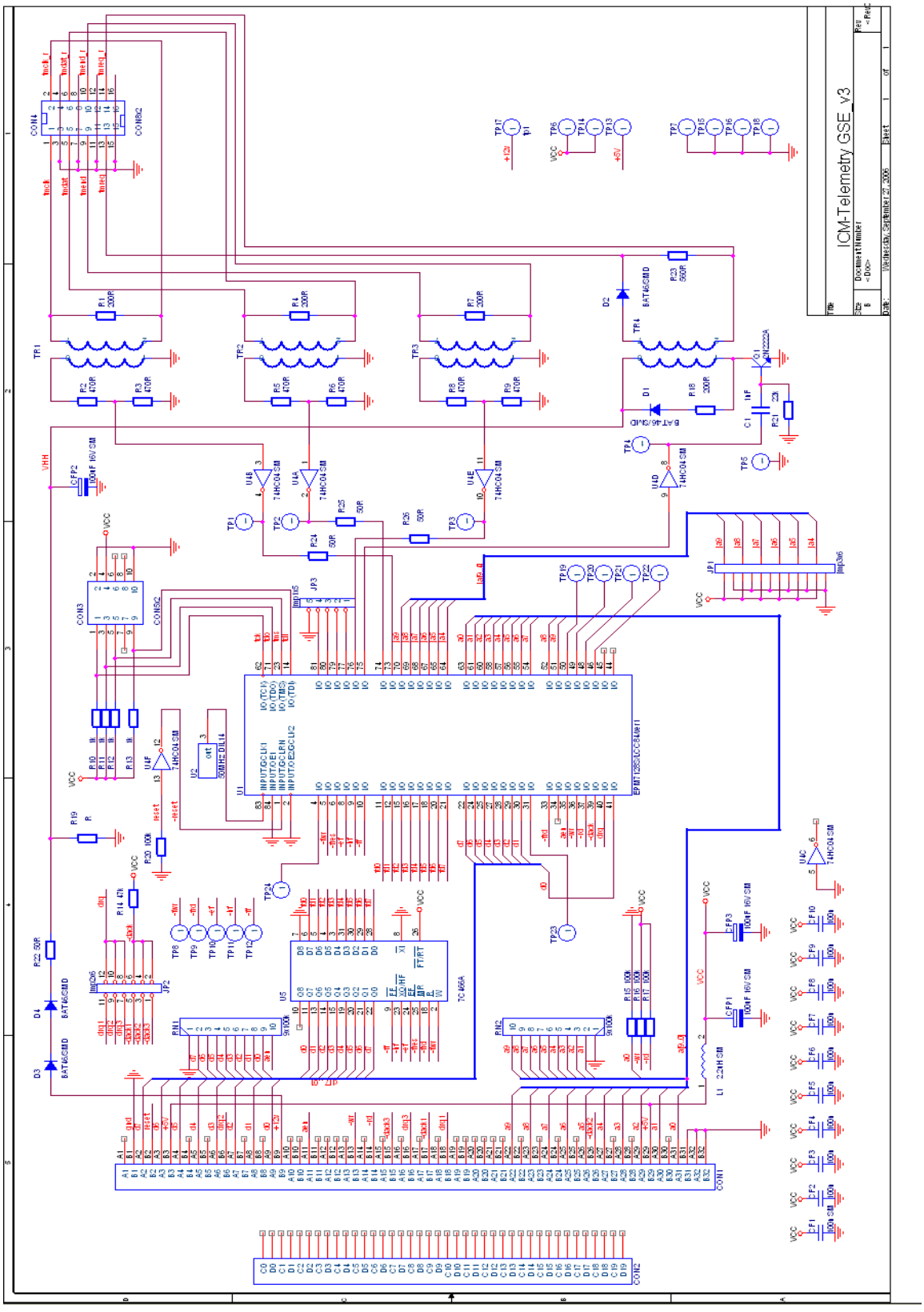


Figure 20 The scheme of BITS Telemetry Test Card

The order of cards in EGSE from the bottom panel to top is:

- PCM-3350 processor card,
- PCM-3618 8-port RS422/485 High-Speed Module; i/o=300h irq5
- PCM-3712 D/A Converter, 2 Channel Analogue Output; i/o=220h, +-10V
- PCM-3718-HG 12 bit DAS Module; i/o=280h
- Bits Telemetry Test Card

5. PWCscript – capture/playback module documentation

5.1. General description

PWCscript is a capture playback module designed for the PWCegse instrument system. The goal of the module is to aid the user through the acquisition process (onboard sequence) by giving the ability to record and playback sequences of user actions performed on the PWCegse graphical user interface. The user action sequences are recorded in a form of XML so it is easily readable by the user and can also be easily processed by computers.

This documentation gives a brief overview of the PWCscript module. In the next section the syntax of the recorded script file and the usage of the module will be discussed. The last section of the document presents the internal working of the module.

5.2. User documentation

5.2.1 Syntax of the PWCscript xml file:

The exact syntax of the xml file used to store the scenario is described by the *pwcscript_v1.dtd* document type definition file.

General tags:
(*loop, include, define_variable, del_variable, set_variable, if, while*)

<loop count=””></loop>

A specific number of iteration over the sequence of sub tags

count: the number of iteration

<include fileURI=””/>

This tag is replaced by the content of the root node in the referenced file

fileURI: path for a valid xml script file to include into the processing sequence. The xml file must be valid

<if statement=””></if>

The sub tags are processed if and only if the value of the *statement* attribute is true

<while statement=””></while>

The sub tags are processed iteratively while the value of the *statement* attribute is true

<define_variable name=”” type=””/>

definition of a variable

name: the name of the variable

type: the type of the variable, possible types are: *STRING*, *INT*, *FLOAT*

<set_variable name=”” value=””/>

set the value of a previously defined variable

name: name of the variable

value: new value

<del_variable name=””/>
delete a variable

Variable reference and arithmetic expressions in the attributes:

$*ref*; *ref* is a previously declared variable name. This reference can be used in any attributes, and it is completely replaced by the actual value of the referenced variable before processing the tag.

$[expr] *expr* is an arithmetic expression build up by variable references, arithmetic operators (+, -, * and /) and numeric or string constants. It is completely replaced by the value of the expression before processing the tag.

Except: *if* and *while* tags:

In this case the whole attribute processed as an expression, so the $[] marking is not necessary. In addition to the four basic arithmetic operators the ‘==’ (equal), ‘!=’ (not equal), ‘>’ (greater than), and ‘<’ (less than) operators can also be used.
(note that the ’<’, ’>’ characters are maintained by the xml parser as tag marking characters!)

Specific tags:

This group contains the tags which describe interaction on the graphic user interface.

<connection act=”” dir=””/>
Establish a connection or disconnect from the server in the specified direction
act: *connection* or *disconnection*
dir: *in*, *out*, *both*

<turnSAS3supply act=””/>
Send a turn on/off SAS3 supply string to the server
act: *on*, *off*

<turnLP1supply act=””/>
Send a turn on/off LP1 supply string to the server
act: *on*, *off*

<turnDP1supply act=””/>
Send a turn on/off DP1supply string to the server
act: *on*, *off*

<turnDFM1supply act=””/>
Send a turn on/off DFM1 supply string to the server
act: *on*, *off*

<turnCWD1supply act=””/>
Send a turn on/off CWD1 supply string to the server
act: *on*, *off*

<turnCORESsupply act=””/>
Send a turn on/off CORES supply string to the server
act: *on*, *off*

<turnLP2supply act=””/>
Send a turn on/off LP2 supply string to the server
act: *on*, *off*

<turnDP2supply act=””/>

Send a turn on/off DP2 supply string to the server
act: *on, off*
<turnRFAsupply act=””/>
Send a turn on/off RFA supply string to the server
act: *on, off*
<turnDFM2supply act=””/>
Send a turn on/off DFM2 supply string to the server
act: *on, off*
<turnCWD2supply act=””/>
Send a turn on/off CWD2 supply string to the server
act: *on, off*
<burstSAS3 act=””/>
Send a turn burst on/off string to the SAS3
act: *on, off*
<burstCORES/>
Send a turn burst on/off string to the CORES
act: *on, off*
<param type=”” value=””/>
type: type of the parameter: sas3, lp1, dp1, lp2, dp2
value: value of the parameter

<delay type=”” value=””/>
type: type of the parameter: day, hour, min, sec
value: value of the parameter
Insert a delay in the command flow before executes the next command

< turnISSvisibility act=””/>
Send an ISS visibility, ethernet turn on/off string to the BSTM
act: *on, off*

<turnAmRvisibility act=””/>
Send an Amateur Radio visibility (connection), turn on/off string to the BSTM
act: *on, off*

<turnBitSvisibility act=””/>
Send an BitSerial connection (visibility), turn on/off string to the BSTM
act: *on, off*

<setCORESmode packets ="" rate=""/>
Send command to set CORES mode
packets: 1,2,3,4,5,6,7
rate: 192, 128, 64 (=N frame/3sec, /375msec, /47msec)

<setTestPattern pattern =""/>
Send command for simulation (to EGSE) of test pattern
pattern: 0, 1, 2, 3, 4
(incr: 0,1,2,…; 0,FF,0,FF,…; 55,66,55,66…; 0,0,0,…; FF,FF,FF,…)

<setAnalogChannels channel1 ="" channel2=""/>
Send command for simulation (to EGSE) to setting the analogue output channels
Channel1: in the range -10.000 …+10.000
Channel2: in the range -10.000 …+10.000

<setSAS3_ADSPcmd value=""/>
Send command to SAS3 ADSP card
value: 0000 …FFFF (any four hex character)

<setSAS3_Generalcmd target = "" value=" "/>
Send command to SAS3 "target" card
target: ADSP, FSA, PC
value: command string, max. 31 characters

Example

In the following example is demonstrated the following activity in XML script form:

1. General information about the XML file, including date of generation
2. PWC script start
    a. *Connect*ion between control PC and embedded processor in *both* (TM and TC direction)
    b. Switch *On* the power supply of the *SAS3* instrument
    c. Wait *4 sec*onds
    d. Send the *800d com*mand to *SAS3* the target is *ADSP* card
    e. Send the *1234* command to *SAS3* the target is the *FSA* card
    f. Send the *4321* command to *SAS3* the target is the *PC* card
    g. Wait *4 sec*onds
    h. Send the *a40d com*mand to *SAS3* the target is *ADSP* card
    i. Wait *6 sec*onds
    j. Burst mode command for SAS3
    k. Switch *Off* the power supply of the *SAS3* instrument
    l. *Disconnect*ion between control PC and embedded processor in *both* (TM and TC direction)
3. PWC script end
4. Prepared by *BSTM team*
5. Comment: Test sequencies

```
<?xml version="1.0" encoding="windows-1250" ?>
<!DOCTYPE PWCscript (View Source for full doctype...)>
<!--  date: 09-21-2006  -->
- <PWCscript>
    <connection act="connect" dir="both" />
    <turnSAS3supply act="on" />
    <delay day="0" hour="0" min="0" sec="4" />
    <setSAS3_ADSPcmd value="800d" />
    <setSAS3_Generalcmd target="FSA" value="1234" />
    <setSAS3_Generalcmd target="PC" value="4321" />
    <delay day="0" hour="0" min="0" sec="4" />
    <setSAS3_ADSPcmd value="a40d" />
    <delay day="0" hour="0" min="0" sec="6" />
    <burstSAS3 />
    <turnSAS3supply act="off" />
    <connection act="disconnect" dir="both" />
  </PWCscript>
  <!-- made by: BSTM team  -->
  <!-- comment: Test sequence  -->
```

5.2.2 The Script Control panel:

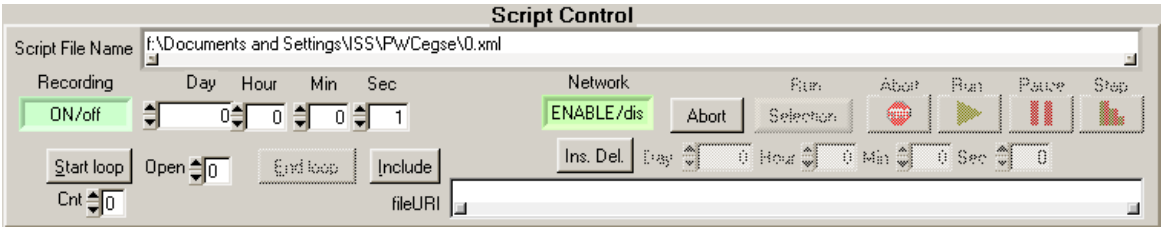


Figure 21 The Script Control panel in Recording Mode

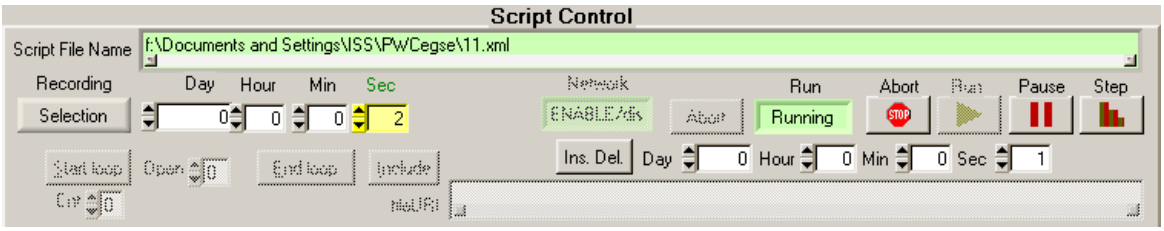


Figure 22 The Script Control panel in Playback mode

The PWCscript module has a graphical user interface which provides the capturing (recording) and playback functions.

Capture

In *capture* state the module writes the user interactions to the specified xml file in the same order as committed by the user on the PWCegse GUI. Capture state starts after pressing the Recording button and selecting the destination file. The *Recording* label on the *Recording* button initiates that the module is in capture state. Stopping the capture mode can be done by the same button.

Playback

In *playback* state the module processes a specified script file. Initiating the playback state can be achieved by the pressing of the Run button and selecting the source script file. The processing of the selected script file can be controlled by the four icon buttons.

Network connection

Disabling the network connection provides the ability to record a script file without sending communication packets to the server. In disabled state the TCP/IP function calls are overloaded with dummy functions to avoid the controlling of the server.

Controlling of the module

Script file name

- in capture mode sets the destination file name
- in playback mode sets the source file name, and the result of automatic validation
    - red: the script failed the validation. Not valid, can not playback
    - green: the script file is valid, ready to start playback

Recording

- *Selection* state: the module is not in capture mode. Pressing the button in this state starts the capturing mode.
- *Recording* state: the module is in capture mode. Pressing the button in this state stops the capturing process.

Delay time

- a *Delay* tag can be inserted into the script with the *Ins Del* button. The value of the delay is given in he *Delay Time* box while the unit of measurement of the delay value can be either *min* (minute) or *sec* (second)
- in *playback* state the *Delay Time* field contains the unexpired time from the last *delay* tag

Network

- sets the state of the network connection and enables control to use the network

Run

- *Selection* state: the module is not in playback mode. Pressing the button in this state starts the playback mode

- *Selected* state: the module is in playback mode. The playback mode can be restarted by pressing the button in this state
- *Running* state: indicates that currently a script file is under processing

Playback control

These control buttons enabled to use only in playback mode.

Abort

- stops and resets the script playing

Run

- starts the playback of the selected script

Pause

- stops but does not reset the playback

Step

- step by step processing of the script

Record control

These buttons are enabled for use only in capture mode.

Start loop

- places a *loop* start tag to the script all the following actions will be placed in the body of the loop until the matching *loop* end tag

Count

- the value of the count attribute in the loop start tag. The value of the numeric box is used when the Start loop button pressed

End loop

- places an end loop tag in the script

Opened

- the number of the opened loop tag in the script file.

Include

- places an include tag to the script

FileURI

- the value of the fileURI attribute in *include* tag. The path and name of a script file to include

5.3 Development documentation

This section gives a brief overview of the internal working and structure of the PWCscript capture playback module. It is structured in two parts. One is a dynamic link library (*DomParserDLL.dll*) containing the main xml processing functions. *DomParserDLL.dll* requires the *xerces-c_2_6.dll* for operation. The other part contains the PWCegse software specific function which are placed and compiled together with the original source code of the software. The interface between the two parts will be also discussed in this section.

5.3.1. DomParserDLL.dll

This library contains the functions corresponding to the script processing. It provides 3 externally callable functions provided to access the functionality. These functions are called *parserInit*, *doParse* and the *parserTeminate*. The *parserInit* function initializes the parser and loads the xml script file while the *parserTeminate* terminates the parser and clears up the memory. The doParse function starts the parsing process and returns only once the process is completed.

The parsing process

In the first step all the script file is read and a document object tree is created from it. The parser starts to walk through the nodes in the tree. As mentioned in the 2 chapter there can be two kind of node in a valid xml. The parser uses the specified function pointers to call the handler functions for all of the nodes corresponding to a specific tag. If an error occurred during the processing then it calls the *ErrorHandler* else it calls first, the *TagStartHandler* and than the *TagEndHandler* functions. The order of node parsing is strictly equivalent with the sequential order of the corresponding tags in the script file. After the *TagEndHandler* is called for the last node in the tree the *doParse* function returns.

5.3.2. Additional functions to PWCegse source

ParseTagStart function

- prototype: int CVICALLBACK *ParseTagStart*(char* Element,
  char* Attributes[],
  int Attr_num)
- implements the *TagStartHandler* function. It calls the tag name specific handler function with the attribute array (*Attributes*[]).

ParseTagEnd function

- prototype: int CVICALLBACK parseTagEnd(char* Element)
- implements the *TagEndHandler* function.

ParseError function

- prototype: int CVICALLBACK parseError(char* ErrorMsg)

- implements the *ErrorHandler* function.

doParseThread function
- prototype: int CVICALLBACK doParseThread(void* ptr)
- this function runs in a new thread. It calls the *doParse* and the *parserTerminate* functions of the *IRFDomParserDLL*.

TurnSupplyTag function
- prototype: int TurnSupplyTag(int iRes, char* Attr[])
- the specific tag start handler for the *turn***supply* tags.

ParamTag function
- prototype: int ParamTag(char* Attr[])
- the specific tag start handler for the *param* tags.

parseDelayTag function
- prototype: void parseDelayTag(char* Attr[])
- the specific tag start handler for the *delay* tags.

parseConnectionTag function
- prototype: void parseConnectionTag(char* Attributes[])
- the specific tag start handler for the *delay* tags.

ScriptSSS function
- prototype:
  int CVICALLBACK ScriptSSS (int panel, int control, int event,
  void *callbackData, int eventData1, int eventData2)
- controls the script playback state. It can *start*, *pause* or *abort* the playback.
  - start
    starts the *doParseThread* function in a new thread if is not already started.
  - pause
    sets the playback state so that the *ParseTagStart* function will not return until the state is not changed.
  - abort
    sets the playback state so that the *ParseTagStart* function will not call the specific handler functions any more.

Validation function
- int Validation(void)
- performs the validation of the selected script file. Calls the parserInit function initialize the file name and the error handler pointer than calls the doParse. If a validation error occurs the error handler function will be called.

PerformSave function
- prototype: void PerformSave(int iCtrl)

- writes the xml tag to the destination file corresponding to the *iCrtl*.

PWC_ClientTCPWrite function

- prototype: int PWC_ClientTCPWrite(unsigned int cHandle,
  void* dPtr, int dSize,
  unsigned int tout)
- overrides the *ClientTCPWrite* function. The *ClientTCPWrite* is called if the Network control is enabled (see 2.2. Network).

PWC_ConnectToTCPServer function

- prototype:
  int PWC_ConnectToTCPServer(unsigned int *conversationHandle,
  unsigned int portNumber, char *serverHostName,
  tcpFuncPtr clientCallbackFunction, void *callbackData,
  unsigned int timeout)
- overrides the *ConnectToTCPServer* function.
  The *ConnectToTCPServer* is called if the Network control is enabled (see 2.2. Network).

PWC_DisconnectFromTCPServer function

- prototype
  int PWC_DisconnectFromTCPServer(unsigned int onversationHandle)
- overrides the *DisconnectFromTCPServer* function.

The *DisconnectFromTCPServer* is called if the Network control is enabled (see 2.2. Network).

5.3.3. IRFDomParserDLL communication interface

The DLL provides 3 externally callable functions for the controlling of the script playback.

parserInit function

- prototype:
  extern int parserInit(vihivStart_type *fpStartHandler* vihivEnd_type *fpTagEndHandler*, vihivError_type *fpErrorHandler*, char* *acFile*)
  - *fpStartHandler*
    Pointer to the tag start handler function in the caller code. The prototype of the function is given by vihivStart_type discussed later in this chapter.
  - *fpTagEndHandler*
    Pointer to the tag end handler function in the caller code. The prototype of the function is given by vihivEnd_type discussed later in this chapter.
  - *fpErrorHandler*

Pointer to the processing error handler function in the caller code. The prototype of the function is given by the vihivError_type discussed later in this chapter.

- *acFile*
  Pointer to an array of chars containing the path to the xml script file to process.

- this function initializes the parser so it must be called first. It sets the 3 call-back function pointer, opens the script file to read and initializes the *xerces* XML DOM parser.

doParse function

- prototype: extern int doParse(void)
- this function call starts the parsing of the specified xml script file. The doParse function returns only when the processing is complete.

parserTerminate function

- prototype: extern int doParse(void)
- this function terminates the parser, and cleans up the memory. It is necessary to call this function after a *doParse* function call.

The caller of the *IRFDomParserDLL* should implement 3 externally callable functions to handle the 3 main events of the script processing as discussed earlier. Only the prototypes of the functions will be presented here.

vihivStart_type

- typedef int (*vihivStart_type)(char* *chTagName*,
  char*[] *chTagAttrList*, int *iAttrNum*)
  - *chTagName*
    pointer to a null-terminated string containing the name of the tag under process.
  - *chTagAttrList*
    pointer to an array of null-terminated strings containing the attribute values of the tag under process. The array contains the values in alphabetical order of the attribute name.
  - *iAttrNum*
    number of the elements in the chTagAttrList.

vihivEnd_type

- typedef int (*vihivEnd_type)(char* chTagName)
  - *chTagName*
    pointer to a null-terminated string containing the name of the tag under process.

vihivError_type

- typedef int (*vihivError_type)(char* chErrorMessage)
  - *chErrorMessage*
    pointer to a null-terminated string containing error messages corresponding to the error occurred during the processing.

6. Decoder files of the housekeeping packets

There can be two types of parameters defined in a decoder file, an enumerated type (*Enum*) or an actual value (*Actual*). Enum types need to have a text description for every possible parameter value. Actual type need to have a base description - either hexadecimal or decimal. There is a possibility to make a hexadcimal dump (*HexDump*) of a certain part from housekeeping packets.

Enum Type Format

[<Field No.>\s<Field Descr.>]\tEnum\t<Word Number>;<Start Bit>,<End Bit>
<Param1>=<Param1 Description>
<Param2>=<Param2 Description>
<Param3>=<Param3 Description>
.....
.....

Actual Type Format

[<Field No.>\s<Field Descr.>]\tActual\t<Word Number>;<Start Bit>,<End Bit>
<Base Type>

There are two commands to control or make more readable the output format in the decoded panel boxes. The Comment insert explanation text, using the control character '^' before Enum or Actual type definitions the automatic "carriage return" and "line feed" characters will be eliminated in the decoded text.

<BaseType>
Base type can be 'Dec', 'Hex', 'Linear' or 'Table'. In Dec or Hex mode the actual value of the selected binary sequence (the binary number between <Start Bit> and <End Bit>) will be printed either in decimal format or in hexadecimal format. In Linear mode a linear interpolation will be executed on the selected binary number. In Table mode the printed value will be selected from a table. The number of the curve uste to calculate linear interpolation must be given after the 'Linear' or 'Table' keywords (eg.: 'Linear 1:' 1 is the number of the curve). The corves must be predefined in a 'table.txt' file. See an example belove for this file

```
TABLE
[Temp1 >1]  (0;0) (1;1) (2;1) (3;2) (4;2) (5;3) (6;3) (7;4) (8;4) (9;5)
      (10;5)      (11;6)      (12;6)      (13;7)      :


[Temp2 >2]  (0;0) (3;5) (6;7) (9;8) (12;8)      (15;15)     (18;17)
      (21;17)     (24;12)     (27;10)     (30;15)     (33;16)
      (36;16)     (39;17)     :
```

HexDump
[<Field No.>\s<Field Descr.>]\tHexDump\t<Start Word Number>;<End Word Number >

Comment
[<Field No.>\s<Text>]\tComment

The meaning of the definition fields:

Field No.
This must start at 1 and be incremented by 1 for each parameter field.
Field Desciptionr
A description of the parameter field
Text
One line text string, which will be copied into the decoded panel box
Word Number
This gives the location of the of the 16 bit word within the HK data. The first word is 0, the second is 1 etc.
Start Bit
This gives the location of the least sign. Bit of the parameter within the selected word.
End Bit
This gives the location of the most sign. Bit of the parameter within the selected word.

Remark:
1. A single bit parameter (e.g. a flag) will have the same value for Start and End bits.
2. The bits in a word should be numbered as follows:

Bit position	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Hexadecimal	1				D				C				9			
Binary	0	0	0	1	1	1	0	1	1	1	0	0	1	0	0	1

In this example, the Hex word 1DC9h is a word as seen on the PC screen from a TM packet received by EGSE if selected the Word order (Menu > Option > Word Order of Data). The result only on the screen will be C91Dh if from the menu selected the Byte order (Menu > Option > Byte Order of Data). The decoding procedure executes word order independently of the screen representation!

Param1
This is the value to which a description will be assigned e.g.
0=Off, 1=On.
This field should only be used if the type is Enum

Param1 Description

This is the assigned description (On or Off in the above example). This field should only be used if the type is Enum!

Base Type

This should contain the word “Hex” or “Dec” to display the parameter in Hexadecimal or Decimal format. This field should only be used if the type is Actual!

Terminator characters

RT	Carriage Return
\s	Space
\t	Tabulator
,	comma
;	semicolon

Example

In the DefCores.txt there are the following lines:

....
[6\sCORES Power1]\tEnum\t0;10,10
0=Off
1=On
[7\sCORES Temperature]\tActual\t0;0,9
Dec
[8\sCORES Command Echo]\tHexDump\t1;6
....

In the example the first word of the housekeeping data the bit 10 indicates a status (On or Off) while the in lower part (0 - 9=10 bits) of the first word is a voltage value which will be visualize in decimal form. This example (definition file) is not real definition of CORES sensor HK data stream!!

The CORES sample HK data stream in words:

0. 079D
1. 1F00
2. 73E4
3. 8911
4. ….
5. …..

The first word

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
					0/**1**	CORES Temperature value (**0x39D**)									

Output result

CORES Power 1 **On**
CORES Temperature **925**
CORES Command Echo
1F00 73E4 8911

Remarks:

1. There is a not real CORES definition file (DefCores.txt)!!!
2. The underlined text is from the sample DefCores.txt <Field Description>
3. The bold text is the decoded information from a sample CORES HK packet using the not real DefCores.txt file
4. The real output will not have underline and bold text, they are only for this explanation