DTUsat

Onboard Architecture

Cecilie M. Larsen
c971744

Gitte M. Nørgaard
c970681

Kim G. Jensen
c971789

Morten Friisgaard
c971403

Søren Hjarlvig
c971483

Tue Becher Jensen
c971651

June 25, 2002, DTUsat Onboard Software Group

Informatics and Mathematical Modelling,
Computer Science and Engineering (CSE)
Technical University of Denmark

Contents

1 Introduction

This project is part of the DTUsat project, which involves more than 50 students at the Technical University of Denmark.
The satellite is based on the CubeSat concept, which has been developed by students and professors at the American University, Stanford. The concept is to make a small satellite in a standard form. The size of the CubeSat is 10x10x10 cm, and it has a maximum weight of 1 kg. Bundles of CubeSats can be sent in orbit together, instead of being sent individually, resulting in a significant reduction of launch costs.

The onboard software is part of the satellite platform and provides basic services for the payload and the other subsystems on the satellite. Those services include:

Communication - Between the different subsystems as well as between the Ground Segment and the satellite.

Scheduling - Execute commands at certain times.

Housekeeping - Collect housekeeping data from the different modules (payload, power, attitude etc.).

Autonomy - Control and error handling when the satellite is out of communication range.

Because of the tight constraints implied by the CubeSat concept, a design decision has been made to equip the DTUsat with only one computer. This means that all the subsystems on the satellite must share the same computer, thus special care must be taken to manage resources and avoid conflicts. As a consequence of this, an important part of the onboard software group's work has been to specify how the software parts of the different subsystems should interact.

The eCos realtime operating system has been chosen as the underlying operating system of the DTUsat software platform. eCos is open source, free of charge, and supports the chosen processor. Furthermore, an emulation tool for Linux are provided, making development and test easier. Due to the lack of an ADA compiler for the hardware platform, the onboard software is written in the C programming language.

This report includes a detailed description of each onboard software module, including an overview of the module design, a description of the module's responsibilities and which commands the module handles.

Reading this report will give a C-programmer the basic skills to design, develop and deploy new modules to the satellite.

2 Onboard Software

In the following sections the architecture of the system will be described. Packet flow through the system will be illustrated and the structure of packets will be shown. Furthermore definitions of important concepts such as modules, commands and time will be given.

The source code for the system can be found at http://cvs.dtusat.dtu.dk. At the time of handing in this report a tag was set in cvs named 'ver1'.

2.1 Architecture

The onboard software consists of the following elements:

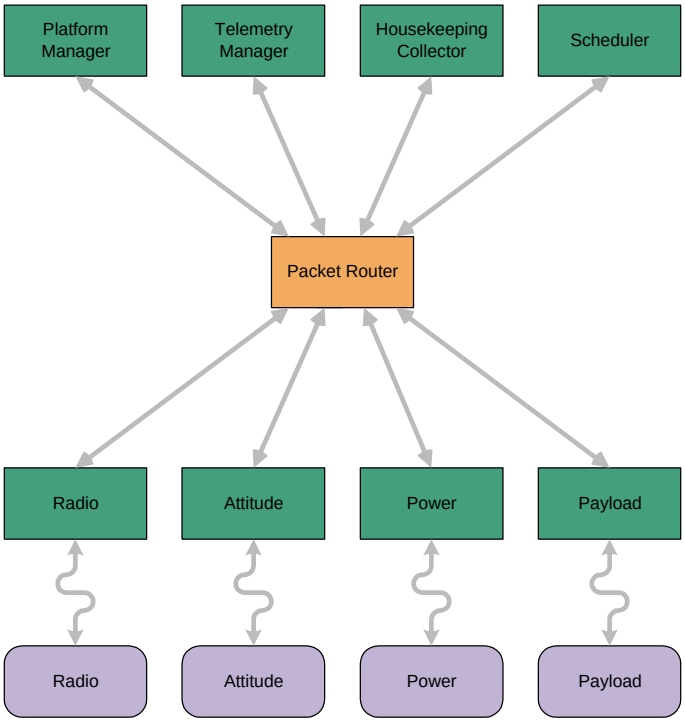

Figure 1: The architecture.

This report focuses on the overall architecture of the onboard software, as well as the modules constituting the basic software platform; the Platform Manager, the Telemetry Manager, the Housekeeping Collector, the Scheduler and the Packet Router. Only the basic skeleton of the other modules is

provided, since we rely on the respective groups to provide the contents of those modules.

Name	**Description**
Platform Manager	Responsible for kicking the hardware watchdog as long as the system works properly. It is also possible to adjust the time on the system through this module.
Telemetry Manager	Stores the packets for the Ground Segment until communication is established.
Housekeeping Collector	Polls each module for housekeeping data.
Scheduler	Stores commands and passes them on to the intended receiver on time.
Packet Router	Routes packets from one module to another.
Radio	Makes it possible to communicate with the Ground Segment using the onboard radio.
Attitude	Controls the attitude subsystem.
Power	Controls the power subsystem.
Payload	The two payloads of the satellite, the camera and the tether, are each controlled by a module.

2.2 The Concept of a Module

A module is an independent part of the system, which operates autonomously and only communicates with the rest of the system through the Packet Router using packets and the communication facilities of the Packet Router.

A module is identified by a number - an ID, which is unique for each module within the system.

Most modules contain several threads - one of these is assigned to receive packets. Other threads could be used for controlling external devices or doing background calculations e.g. attitude calculations or image compression.

Each module is required to implement certain functions and commands, please refer to section 3 on page 15 for a detailed description of the requirements.

2.3 The Concept of Telecommands

The Ground Segment and the modules, are able to communicate using commands and confirmations. Commands and confirmations can also be used for inter-module communication.

A command is a request to a module to do something. For each command it must be defined if it takes any arguments and whether it returns a confirmation. If it does, the possible return values must be defined as well. All telecommands used directly by the Ground Segment must return a confirmation.

All commands have a sequence number, the original command and a confirmation is paired together using this number.

Each command is identified with a number and 10 numbers are reserved for each module. The numbers 0-9 are reserved for global commands.
If a module needs more than 10 commands, the command numbers 100-255 are free to use, but it should be coordinated with the other module groups to keep the command numbers unique.

Each error code in a confirmation is also identified by a number. An error code in a confirmation corresponding to a command must have a unique number within that command, thus two error codes can be identical as long as they correspond to two different commands. To determine what kind of confirmation is received, the sequence number has to be compared with the sequence numbers of the issued commands.

2.4 The Concept of Onboard Time

The onboard time, which can be obtained by a module by invoking the getCurrentTime function, is measured in milliseconds since midnight, January 1, 1970. A set of telecommands used for adjusting the onboard time, is provided by the Platform Manager.

eCos maintains an internal real-time clock. When the time is set, using the PLM_SET_TIME telecommand, the internal clock remains unaltered. This means that if a thread waits e.g. for a semaphore with a timeout of 10 seconds, the thread will wait at least 10 seconds if the semaphore is not signalled, even though the time might be adjusted while it waits. The absolute time can be adjusted through the Platform Manager. This is described in details in section 4.

The onboard time runs, even though it has not been set from the Ground Segment. Whenever the system reboots, the time will be reset, and the satellite's current time will be the time since restart.

Since the clock of the onboard computer drifts due to changes in temperature etc., the first telecommand in a communication session should be a time adjustment command.

If a module depends on having a correct absolute time, e.g. the Attitude module, it is recommended to check if the time has been set, before trusting it. This could be done by checking if the current time is before or after the launch date.

A possible extension could be to store the current time in the flash memory, with certain intervals, during the normal operation of the system, this would make is possible to obtain an approximation of the absolute time after a reboot.

2.5 Packet Format

For the communication between the modules on the satellite and between the satellite and the Ground Segment, a standard packet format has been defined.

2.5.1 Description

A packet consists of four different fields; the first three fields constitute the header of the packet; the last field MsgData can be of three different formats depending on the type of the packet.

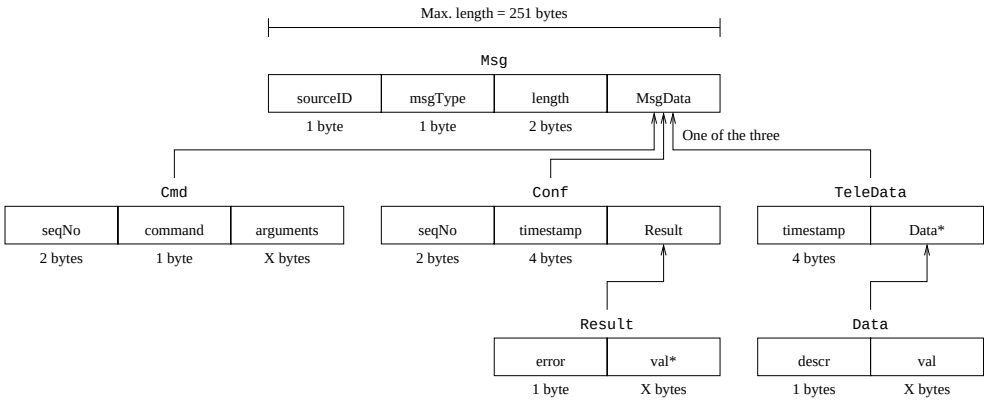

Figure 2: Packet format.

It is assumed that the maximum length of a packet is 251 bytes, but it can be modified later, depending on the requirements of the Radio module.

An easy way to create packets is to use the DTUsat library functions:

- createCmdPacket
- createConfPacket
- createTeledataPacket

A packet is used for communication and it contains:

Name	Description	Type
sourceID	The id of the recipient module; describing which module the packet should be sent to.	A module ID.
msgType	The type of the packet and thereby the type of MsgData.	One of MSG-TYPE_CMD, MSG-TYPE_CONF or MSG-TYPE_TELEDATA
length	The length of the field MsgData.	
MsgData	The actual data/message being sent.	One of Cmd, Conf or TeleData.

Command (Cmd) is used for sending a command to a module, all commands are defined in the DTUsat library:

Name	Description	Type
seqNo	The sequence number of the command packet.	
command	The command invoked in the receiving module.	A command.
arguments	The arguments to the command; there can be several arguments.	

Confirmation (Conf) is used for sending a confirmation responding upon a received command:

Name	Description	Type
seqNo	The sequence number of the corresponding command packet.	
timestamp	A timestamp describing when this packet was made.	A four bytes integer containing seconds.[1]
error	An error code describing the type of error.	
val	Relevant values to the error; there can be several values.	

Telemetry data (TeleData) is used for sending information to Ground

[1] For further information about time see section 2.4.

Segment:

Name	Description	Type
timestamp	A timestamp describing when this packet was made.	A four bytes integer containing seconds.
descr	A description of the following value, there can be several of these pairs of descr and val.	
val	Some kind of value.	

2.5.2 Integer Format

All integers, 2 and 4 bytes, must be written in little Endian. The DTUsat library contains methods for writing and reading integers in the proper form. This is important to remember when writing/reading timestamps, lengths, sequence numbers and other values like that.

2.5.3 Packet Priorities

Packets sent to Ground Segment can be sent with different priorities. Highest priority packets are sent first, and lowest priority packets are deleted first in case of no free space. The oldest packets with the lowest priority are deleted first.
If a confirmation packet is sent to Ground Segment without a priority a default value is used, the default value can be changed using the Platform Manager.
We recommend to use priorities as stated below:

PRIO_LOW For standard telemetry data.

PRIO_MIDDLE For confirmations.

PRIO_HIGH For telemetry data and confirmations exceeding the routine.

2.5.4 Sequence Numbers

Sequence numbers are used to bind a confirmation packet to a command packet.
Commands sent from Ground Segment has the first bit set, and commands created on the satellite has not, so it is always possible to determine whether

the command originated from Ground Segment or from another module on the satellite.

2.5.5 Packet Syntax

Syntax of the packet format:

```
Msg       :: sourceID, msgType, length, MsgData
MsgData   = Cmd | Conf | TeleData
Cmd       :: seqNo, command, arguments
Conf      :: seqNo, timestamp, Result
Result    :: error, val*
TeleData  :: timestamp, Data*
Data      :: descr, val
```

2.6 Packet Router

The Packet Router is a library that facilitates communication between the modules and between a module and the Ground Segment.
It is responsible for sending packets from one module to another, and has to store the packets for a short while, until the receiving module is ready to get the packet.

2.6.1 Description

The Packet Router provides three important facilities, which are used by every module:

- sendMsg
- sendMsgToEarth
- getMsg

The Packet Router maintains a mailbox for each module, each mailbox can be adjusted in size according to the needs of the module.

When a module wishes to send a packet to another module or to the Ground Segment it delivers it to the Packet Router along with a receiver ID or a priority.
If there is room the packet is added to the receiving module's mailbox otherwise an error is reported to the sending module and in this case it must decide what to do.

When a module receives a message it must handle the message in a way so it can always be guarantied that a CMD_ALIVE can be received and handled in the allowed time frame.

When calling getMsg the thread blocks until a message is ready in the mailbox.

When sending a message one can call either sendMsg or sendMsgToEarth. When calling sendMsg one must supply a receiver ID. If the ID is GND and the packet is a confirmation message it will be given DEFAULT_PRIORITY[2], if it is not a confirmation message it will be given PRIO_LOW. sendMsgToEarth allows the sending module to decide the priority.

[2]DEFAULT_PRIORITY can be set through the Platform Manager.

2.7 Packet Flow

In the following sections the packet flow will be described, in order to give an overview of the system infrastructure and how the modules interact.

2.7.1 Communication from Ground Segment to Satellite

When the Ground Segment issues a command to a module on the satellite the following elements of the onboard software are involved:

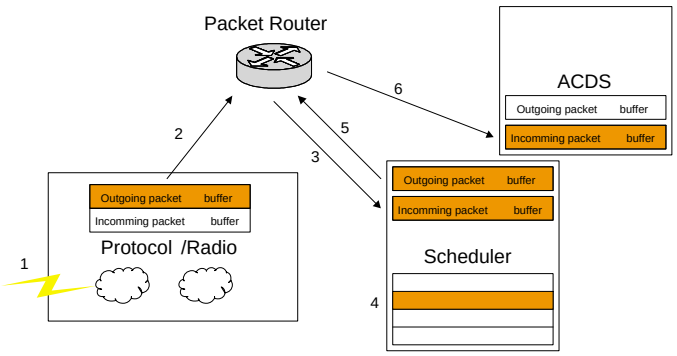

Figure 3: Communication from Ground Segment.

In the following example the Ground Segment wishes to send a command to the Attitude module. The command should not be executed right away, but at some later point in time. This is accomplished by wrapping the command packet to the Attitude module in another command packet, instructing the Scheduler module to send the inner command packet at the intended time.

The Radio module receives the command from Ground Segment, which it puts in its outbox, in order to send it to the Scheduler module. By the use of the Packet Router, the message is moved from the Radio module's outbox to the Scheduler module's inbox. The Scheduler stores the packet until it should be executed. When the time is up, the packet is put into the outbox in order to be delivered to the intended module. The module then receives the packet, again by using the Packet Router, and should now carry out the command sent from Ground Segmemt.

2.7.2 Communication from Satellite to Ground Segment

When a module wishes to send data to the Ground Segment the following elements are involved:

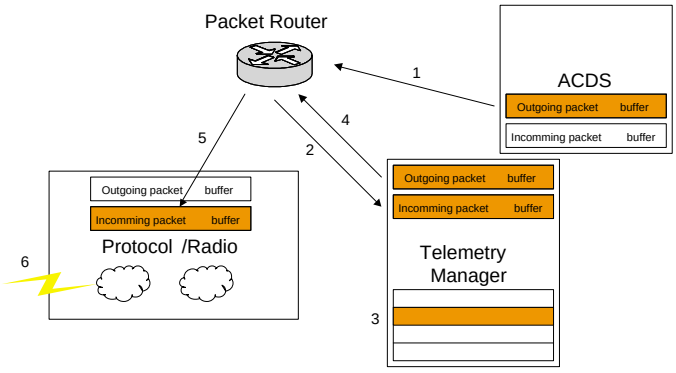

Figure 4: Communication to Ground Segment.

The module creates a packet containing the data intended for the Ground Segment. The packet is put into the module's outbox and by the use of the Packet Router the message is sent to the Telemetry Manager module. The Telemetry Manager module receives the packet in its inbox and stores it until radio contact with the Ground Segment is established. When the radio contact is established, the Telemetry Manager module sends the packet to the Radio module using the Packet Router. Finally the Radio module will transmit the packet to Ground Segment.

3 Developing Modules

This section presents a set of guidelines for module development. The individual groups should follow these guidelines in order to ensure correct operation of the system.

The guidelines are formulated as a help for the module developers during design and implementation of their part of the system. Each group developing modules is expected to hand in a small report documenting their module. The report should describe the telecommands the module implements and how the guidelines are realized in the implementation.

3.1 Overall Behavior

- The modules are not allowed to interfere with each other, only to communicate with each other through the Packet Router, using packets and the communication facilities of the Packet Router.

- A module is expected to empty its inbox regularly using the getMsg function of the Packet Router. Regularly is not further defined but a module must be able to answer an alive command within a certain period of time determined by the global constant MAXTIMESINCEACK. Hence the module must be able to handle and process a packet within this time and be ready for a new packet.

- A module is expected to answer an alive command immediately i.e. within the period of time MAXTIMESINCEACK.

- A module is not allowed to allocate memory dynamically, only static memory allocation within its assigned memory is allowed.

- A module is not allowed to address memory outside its assigned memory.

- It is expected that the facilities of eCos are used carefully, some of the allowed facilities are: semaphores, mutexes, timers, conditions and threads.

- A module is expected to keep track of whether or not its threads are alive.

An example of these requirements can be found in the appendix D.

3.2 Naming Conventions

Since all the modules share the same name space, the following conventions must be observed when declaring global variables and functions: All function names must have the form '*prefix_functionname*', global variables should be named correspondingly '*prefix_ variablename*'. E.g. the names 'pow_getBattvoltage()' and 'pow_battvoltage' could be used in the power module.

ID	Prefix	Description
0	plm	Platform Manager
1	sch	Scheduler
2	hoc	Housekeeping Collector
3	tel	Telemetry Manager
4	rad	Radio
5	att	Attitude
6	pow	Power
7	cam	Camera
8	tet	Tether
50	gnd	Ground Segment

3.3 Using the Packet Router

A priority marks the packets, which are to be sent to Ground Segment, saying how important it is, that this packet actually reaches Ground Segment. The possible priorities are PRIO_HIGH, PRIO_MIDDLE and PRIO_LOW.

The packets has a standard size, MAX_PACKETSIZE, which is the amount of memory that should be allocated, when a packet is to be received, and it is the maximum size of a packet, which is to be sent. If a module needs to send data bigger than this, the data has to be divided and sent in several packets.

To receive a packet a module should call the function getMsg of the Packet Router with its ID and a pointer to a specific address of its memory as arguments. Since the packets distributed by the Packet Router has a standard size, the module needs to have enough memory for the packet before invoking getMsg. This memory must be pre-allocated.

To send a packet, the module should call the function sendMsg with the pointer to the packet, which is to be sent, and the ID of the recipient as arguments. If the packet is to be sent to the Ground Segment with a specific priority the function sendMsgToEarth can be used.

As mentioned earlier there are some functions in DTUsat library which makes it easier to create packets. When creating a packet for housekeeping data the function createTeledataPacket should be use. When a command packet is created the function getNewSeqNo is used to make a sequence number which uniquely identifies the command. The sequence number is given as one of the arguments to the function createCmdPacket. Creating a confirmation packet is done by using createConfPacket with the sequence number of the corresponding command as one of the arguments.

3.4 Required Functions

A module is required to implement the following variables and functions:

- Thread *prefix*_packetreceiver
- int *prefix*_init()
- int *prefix*_destroy()

The thread *prefix*_packetreceiver should be devoted to receive packets from the Packet Router.

The functions *prefix*_init and *prefix*_destroy will be called in order to respectively start and stop the module when the system is initialized, reset and possibly stopped during a safe mode period.

The function *prefix*_init should initialize all variables and threads, which are to be used in the module and the threads (by choice) are started.

The function *prefix*_destroy should kill all (running) threads.

3.5 Required Telecommands

All modules must implement the two following telecommands:

- CMD_ALIVE
- CMD_HKDATA

CMD_ALIVE is sent by Platform Manager, which uses each modules reply to check if they are running. If a modul does not reply to a CMD_ALIVE, the Platform Manager will not kick the watchdog and eventually the system

will reboot. If a module replies with a CONFIRM_ALIVE_NACK it means that one or more threads in the modules do not work properly.
CMD_HKDATA is sent from Housekeeping Collector to make each module send a status message to Ground Segment.

3.6 Budgets

Since only static memory allocation is used, the compiler can calculate the exact memory usage. When developing the modules it should be kept in mind that memory is a limited resource. It is important to know the runtime and the processor usage of the different telecommands in worstcase scenarios. The same must be known for background tasks. In addition the deadline for the different tasks must be known. This information is needed to ensure that is it feasible to operate the satellite.

It is difficult to predict the runtimes before we get the actual platform running, so until then just keep in mind that processor time also is a limited resource.

4 Platform Manager (PLM)

The Platform Manager has two main responsibilities:

- Kicking the hardware watchdog.
- Maintaining the satellite time.

4.1 Solution

The following two subsections describe how the PLM module's responsibilities are handled.

4.1.1 Watchdog Functionality

The hardware watchdog must be kicked regularly in order to avoid hard reset of the satellite.

All modules of the satellite shall be able to determine whether they are fully functioning or have encountered serious execution problems.

The PLM module's main task is to check that all modules on the satellite are fully functioning and then kick the hardware watchdog. If any of the vital satellite modules are not fully functioning, the PLM module stops kicking the hardware watchdog - which in return will result in a reboot of the satellite onboard computer.

The PLM module determines the status of the satellite by sending `CMD_ALIVE` commands to all other modules on the satellite. Each module must then return a confirmation, stating the status of the specific module. When the PLM module receives such confirmations, it records the eCos system time for the specific module. If the PLM module has not received confirmations from a module for a predetermined period of time, it will stop kicking the hardware watchdog.

4.1.2 Satellite Time Adjustment

In order to have real time available to the attitude calculations, a time-adjustment variable is maintained. The satellite real time is based on the eCos system's internal clock. It is assumed that the internal system clock has millisecond resolution.

The PLM module is responsible for making changes to the time-adjustment variable. The PLM module offers 3 different methods to adjust the time. One method is a command called `PLM_SET_TIME_CMD`, which given a number of seconds and milliseconds, sets the satellite real time by changing the time-adjustment variable. The two other methods are commands to increase or decrease the time-adjustment variable by a given number of milliseconds.

The time may be unknown if it has not been set yet, in that case it will not make any sense to use the methods for increasing or decreasing the time.

4.2 Housekeeping

The following table describes possible housekeeping values.

DescrID	Description	Value
0	The time (in seconds) of the oldest alive acknowledgement.	4 bytes unsigned int
1	The time-adjustment variable (in seconds, milliseconds).	4 bytes unsigned int, 4 bytes unsigned int
2	The number of received nacks.	4 bytes unsigned int

4.3 Telecommands

The following subsections describe the telecommands accepted by the PLM module

4.3.1 `CMD_HKDATA`

Description: Housekeeping data is generated upon request from the HOC module.
Arguments: Yes, the following:

Description	Type
The priority of the TeleData packet containing the housekeeping data.	1 byte char

Returns confirmation: Yes, the following:

Error code	Description	Values
0	Housekeeping data generated successfully.	

4.3.2 PLM_SET_TIME_CMD

Description: The time-adjustment variable is changed, so the real time corresponds to a given time described by seconds and milliseconds.
Arguments: Yes, in the following order:

Description	Type
New real time in seconds (floored).	4 byte unsigned int
Remaining milliseconds.	4 byte unsigned int

Returns confirmation: Yes, the following:

Error code	Description	Values
0	The real time of the satellite has been updated successfully.	

4.3.3 PLM_INC_TIME_CMD

Description: The time-adjustment variable is increased.
Arguments: Yes, the following:

Description	Type
The real time is to be increased with this time given in milliseconds.	4 byte unsigned int

Returns confirmation: Yes, the following:

Error code	Description	Values
0	The real time of the satellite has been increased successfully.	

4.3.4 PLM_DEC_TIME_CMD

Description: The time-adjustment variable is decreased.
Arguments: Yes, the following:

Description	Type
The real time is to be decreased with this time given in milliseconds.	4 byte unsigned int

Returns confirmation: Yes, the following:

Error code	Description	Values
0	The real time of the satellite has been decreased successfully.	

4.3.5 PLM_SET_CONFIRM_PRIO

Description: The default priority on confirms are changed.
Arguments: Yes, the following:

Description	Type
The new default priority.	char

Returns confirmation: Yes, the following:

Error code	Description	Values
0	The default confirmation priority has been changed.	

5 Scheduler (SCH)

The purpose of the Scheduler is to store commands that are to be executed later on, at a given time.

5.1 Solution

The SCH module maintains a list of packets along with the recipient of the packet and the time of its execution. When the time denoted for a packet is passed, the packet is delivered to the recipient using the Packet Router.

5.1.1 Behaviour

The Scheduler will at any time keep a reference to the next packet that is to be delivered. The Scheduler sleeps until the next packet is to be delivered, unless a delete or add command is received. The sleep time will be recalculated if a delete or add command is received.

5.1.2 Availability

The Scheduler offers a command to add packets to be scheduled for delivery to a given recipient at a given time. If there is no more space available to store the packets, the sender will be informed, but the packet will be lost. As well as offering an add command, the Scheduler offers a delete command which removes all scheduled deliveries within a given time interval.

5.1.3 Time

If the time is changed while packets are enqueued, the Scheduler will send all packets, which have a timestamp less than the satellites new time. Hence packets scheduled with different times, can be sent at the same time.
Packets can be scheduled even though the time is not set, but if Earth synchronized time is used in the packets it will take very long time before the packets are actually scheduled.

5.2 Housekeeping

The following table describes the housekeeping values.

DescrID	Description	Value
0	Number of packets currently to be scheduled.	4 bytes unsigned int
1	The scheduled time for the next packet to be delivered.	4 bytes unsigned int
2	The sequence number for the next packet to be delivered.	2 bytes unsigned int

5.3 Telecommands

The following subsections describe the telecommands accepted by the SCH module.

5.3.1 `CMD_ALIVE`

Description: Used by the Platform Manager to query about the status of the module.
Arguments: None.
Returns confirmation: Yes, one of the following:

Error code	Description	Values
0	I'm alive.	
1	Some of my threads are not alive.	

5.3.2 `CMD_HKDATA`

Description: Housekeeping data is generated upon request from the HOC module.
Arguments: Yes, the following:

Description	Type
The priority of the TeleData packet containing the housekeeping data.	1 byte char

Returns confirmation: Yes, the following:

Error code	Description	Values
0	Housekeeping data generated successfully.	

5.3.3 SCH_ADD

Description: A new packet to be scheduled is added. If the time is overdue the packet is executed right away.
Arguments: Yes, in the following order:

Description	Type
Id of the recipient.	1 byte char
The time (in seconds) of the delivering.	4 byte unsigned int
The packet to be delivered.	x byte char array

Returns confirmation: Yes, one of the following:

Error code	Description	Values
0	New packet scheduled successfully.	
1	Scheduling of packet failed because of lack of space.	

5.3.4 SCH_DEL

Description: Scheduled packets within a given time-interval are deleted.
Arguments: Yes, in the following order:

Description	Type
The start-time (in seconds) of the interval.	4 byte unsigned int
The end-time (in seconds) of the interval.	4 byte unsigned int

Returns confirmation: Yes, the following:

Error code	Description	Values
0	Number of packets deleted.	4 byte unsigned int

6 Housekeeping Collector (HOC)

The purpose of the Housekeeping Collector is to make it possible to get housekeeping data on regularly basis.

6.1 Solution

The HOC module polls the different modules for housekeeping data with certain periods of time; providing a simple and efficient way of collecting housekeeping data from all the modules of the satellite.

6.1.1 Housekeeping Data

When a module is requested to send some housekeeping data to Ground Segment, the data have to contain some selected status information about the module and the physical subsystem controlled by the module (if any). It is up to each module to determine, which information are useful. E.g. the Power module could provide information about:

- Battery voltage
- Battery temperature
- Solar panel voltage
- Solar panel temperature
- Power consumed by different subsystems
- Etc.

6.1.2 Logging Periods

It is possible to have two logging periods for the same module; this could be used to ensure that we receive housekeeping data from a module with at least some period of time and - if there is enough storage space and transmission time available - more often. E.g. you could tell the HOC module to request a high priority status packet from a module every hour and a low priority status packet every 5 minutes.
If there is not enough space available for a new logging period to be inserted the sender of the request will be informed and the request will be lost.

As a default all modules will have a logging period defined by the global constant HOC_DEFAULT_LOG_PERIOD and telemetry is to be sent to Ground Segment with low priority.

6.1.3 Time

If the time is changed an irregularity in polling housekeeping data from the different modules will occur, as the time a housekeeping command is to be sent depends upon the last time the command was sent and the logging period.
If the time is increased housekeeping data will be polled prior than intended.
If the time is decreased housekeeping data will be polled perhaps much later than intended.
When the irregularity of polling the first housekeeping data from every module after the time is changed is passed, the HOC module will proceed as planed.

6.2 Housekeeping

The following table describes possible housekeeping values. All of the following values are reset after sending housekeeping data.

DescrID	Description	Value
0	Total number of housekeeping commands issued.	4 byte unsigned int
1	Total number of housekeeping commands confirmed.	4 byte unsigned int
2	Number of housekeeping commands sent to PLM.	4 byte unsigned int
3	Number of housekeeping commands confirmed by PLM.	4 byte unsigned int
4	Number of housekeeping commands sent to SCH.	4 byte unsigned int
5	Number of housekeeping commands confirmed by SCH.	4 byte unsigned int
6	Number of housekeeping commands sent to HOC.	4 byte unsigned int
7	Number of housekeeping commands confirmed by HOC.	4 byte unsigned int
8	Number of housekeeping commands sent to TEL.	4 byte unsigned int
9	Number of housekeeping commands confirmed by TEL.	4 byte unsigned int
10	Number of housekeeping commands sent to RAD.	4 byte unsigned int
11	Number of housekeeping commands confirmed by RAD.	4 byte unsigned int
12	Number of housekeeping commands sent to ATT.	4 byte unsigned int
13	Number of housekeeping commands confirmed by ATT.	4 byte unsigned int
14	Number of housekeeping commands sent to POW.	4 byte unsigned int
15	Number of housekeeping commands confirmed by POW.	4 byte unsigned int
16	Number of housekeeping commands sent to CAM.	4 byte unsigned int

continued on next page..

17	Number of housekeeping commands confirmed by CAM.	4 byte unsigned int
18	Number of housekeeping commands sent to TET.	4 byte unsigned int
19	Number of housekeeping commands confirmed by TET.	4 byte unsigned int

6.3 Telecommands

The following subsections describe the telecommands accepted by the HOC module

6.3.1 `CMD_ALIVE`

Description: Used by the Platform Manager to query about the status of the module.
Arguments: None
Returns confirmation: Yes, one of the following:

Error code	Description	Values
0	I'm alive.	
1	Some of my threads are not alive.	

6.3.2 `CMD_HKDATA`

Description: Housekeeping data is generated upon request from the HOC module.
Arguments: Yes, the following:

Description	Type
The priority of the TeleData packet containing the housekeeping data.	1 byte char

Returns confirmation: Yes, the following:

Error code	Description	Values
0	Housekeeping data generated successfully.	

6.3.3 HOC_INSERT_PERIOD

Description: A logging period for a module with a certain priority is inserted. The first housekeeping command is sent at the time of the insert command plus the period.
Arguments: Yes, in the following order:

Description	Type
The module to get housekeeping data from.	1 byte char
The period (in seconds) determining the time between getting housekeeping data.	4 byte unsigned int
The priority of the packet with housekeeping data.	1 byte char

Returns confirmation: Yes, one of the following:

Error code	Description	Values
0	Period inserted successfully.	
1	Failed to insert period.	

6.3.4 HOC_DELETE_PERIOD

Description: A logging period for a module is removed.
Arguments: Yes, in the following order:

Description	Type
The module to remove a logging period from.	1 byte char
The period (in seconds) identifing the exact logging to be removed.	4 byte unsigned int

Returns confirmation: Yes, one of the following:

Error code	Description	Values
0	Period deleted successfully.	
1	Failed to delete period.	

6.3.5 HOC_DELETE_MODULE_PERIODS

Description: All logging periods for a module are removed.
Arguments: Yes, the following:

Description	Type
The module to remove all periods from.	1 byte char

Returns confirmation: Yes, the following:

Error code	Description	Values
0	All periods for module deleted.	

6.3.6 `HOC_DELETE_ALL_PERIODS`

Description: All logging periods are removed. It can be used to reset the queue if something seems wrong.
Arguments: None.

Returns confirmation: Yes, the following:

Error code	Description	Values
0	All periods deleted.	

7 Telemetry Manager (TEL)

The Telemetry Manager handles the storage of telemetry.
Since the satellite is out of communication range most of the time, it is necessary to store the telemetry generated by the modules until radio communication can be established. When radio contact is made with the Ground Segment, the most important data should be transmitted first.

7.1 Solution

The telemetry is stored in a central data structure maintained by the TEL module. As a result of this, the individual modules do not have to worry about allocating and administering memory for long term storage of telemetry. Furthermore, the centralized approach makes it possible to prioritise all telemetry in a well-considered and consistent way. E.g. if the Telemetry Manager runs out of storage space, before communication with the Ground Segment is established, it is possible to delete lower priority packets in favour of new packets with higher priority. Another advantage is the transparency - a module does not have to worry about whether it is sending data to another module or to the Ground Segment.

7.1.1 Adding Policy

If free space is available all packets are accepted. If not, the oldest packet with the lowest priority is deleted, but only if the packet to be inserted has higher or equal priority compared to the packet with the lowest priority already in the queue. This result in a new packet, with a priority equal to the priority of the packet with the lowest priority on the queue, is accepted and the old packet is overwritten.

7.1.2 Transmission Policy

The newest packet, with the highest priority is transmitted first.

7.2 Packet Flow during a Communication Session

When communication is established the Radio module issues a TEL_GET_NEXT_TO_TRANSMIT command, the Telemetry Manager returns a confirmation packet containing the packet to transmit. If the

packet is transmitted successfully to Ground Segment, the Radio module issues a
TEL_CONFIRM_DELIVEY command, otherwise, if the transmission failed, a TEL_NO_DELIVERY command is issued. In both cases the argument should contain the sequence number of the original
TEL_GET_NEXT_TO_TRANSMIT command. When the communication session is terminated, all packets not confirmed, can be marked as ready for transmission again using the TEL_CONNECTION_CLOSED.

7.3 Housekeeping

The following table describes the housekeeping data.

DescrID	Description	Value
0	Total number of packets on queue.	4 byte unsigned int
1	Number of PRIO_LOW packets on queue.	4 byte unsigned int
2	Number of PRIO_MIDDLE packets on queue.	4 byte unsigned int
3	Number of PRIO_HIGH packets on queue.	4 byte unsigned int
4	Total number of discarded packets, since last reboot.	4 byte unsigned int
5	Number of discarded PRIO_LOW packets, since last reboot.	4 byte unsigned int
6	Number of discarded PRIO_MIDDLE packets, since last reboot.	4 byte unsigned int
7	Number of discarded PRIO_HIGH packets, since last reboot.	4 byte unsigned int
8	Number of packets on queue from PLM.	4 byte unsigned int
9	Number of packets on queue from SCH.	4 byte unsigned int
10	Number of packets on queue from HOC.	4 byte unsigned int
11	Number of packets on queue from TEL.	4 byte unsigned int
12	Number of packets on queue from RAD.	4 byte unsigned int
13	Number of packets on queue from ATT.	4 byte unsigned int
14	Number of packets on queue from POW.	4 byte unsigned int
15	Number of packets on queue from CAM.	4 byte unsigned int
16	Number of packets on queue from TET.	4 byte unsigned int

7.4 Telecommands

The following subsections describe the telecommands accepted by the TEL module.

7.4.1 `CMD_ALIVE`

Description: Used by the Platform Manager to query about the status of the module.
Arguments: None.
Returns confirmation: Yes, one of the following:

Error code	Description	Values
0	I'm alive.	
1	Some of my theads are not alive.	

7.4.2 `CMD_HKDATA`

Description: Housekeeping data is generated upon request from the HOC module.
Arguments: Yes, the following:

Description	Type
The priority of the TeleData packet containing the housekeeping data.	1 byte char

Returns confirmation: Yes, the following:

Error code	Description	Values
0	Housekeeping data generated successfully.	

7.4.3 `TEL_ADD_TO_QUEUE`

Description: A packet is inserted into the internally maintained datastructure. This command is primarily used by the sendMsgToEarth function.
Arguments: Yes, in the following order:

Description	Type
Priority.	1 byte char
Packet.	x byte char array

Returns confirmation: No.

7.4.4 TEL_GET_NEXT_TO_TRANSMIT

Description: This command is invoked by the Radio module when it is ready to transmit to the Ground Segment.
Arguments: None.
Returns confirmation: Yes, one of the following:

Error code	Description	Values
0	Packet to transmit.	x byte char array
1	Nothing to transmit.	

7.4.5 TEL_CONFIRM_DELIVERY

Description: This command is invoked by the Radio module when it successfully has transmitted a packet to the Ground Segment meaning that the Telemetry Manager can delete the packet from its storage.
Arguments:

Description	Type
Command sequence number from a previous TEL_GET_NEXT_TO_TRANSMIT command.	2 byte unsigned short

Returns confirmation: No.

7.4.6 TEL_NO_DELIVERY

Description: This command is invoked by the Radio module when it has failed to transmit a packet to the Ground Segment meaning that the Telemetry Manager cannot delete the packet from memory and the packet is marked as ready for transmission again.
Arguments:

Description	Type
Command sequence number from a previous TEL_GET_NEXT_TO_TRANSMIT command.	2 byte unsigned short

Returns confirmation: No.

7.4.7 TEL_CONNECTION_CLOSED

Description: This command is invoked by the Radio module when it no longer has contact to the Ground Segment. The packets transmitted and

not yet confirmed is marked as ready for transmission again.
Arguments: None.
Returns confirmation: No.

7.4.8 TEL_DELETE

Description: Packets on the queue are deleted. It can be used to reset the queue if something seems wrong.
Arguments:

Description	Type
The delete type.	1 byte char
Delete argument.	1 byte char (delete type 1 or 4) or 4 byte unsigned int (delete type 2 or 3)

Allowed delete types are:

0 : All.

1 : All packets from module given in arg.

2 : All packets older than timestamp given in arg.

3 : All packets newer that timestamp given in arg.

4 : All packets with priority given in arg.

Returns confirmation: Yes, the following:

Error code	Description	Values
0	Number of packets deleted.	4 byte unsigned int

8 To do

In order to make the onboard software work properly on the onboard computer there are some things still missing.

8.1 Integration

First of all the software has to be tested systematically, that could include describing the different scenarios and making the tests to assure that the software works as intended.
During the development of the software it has been tested on the onboard computer, but problems could occur when the software finally is migrated to the onboard computer.

As the rest of the modules are implemented, they are to be integrated with the onboard software platform as well as the onboard computer.

8.2 Missing Platform Features

The destroy function in the modules has not been implemented yet. This would be obvious to do and at the same time test how threads in eCos react as they are destroyed and whether it is possible to initialise them again.

Furthermore the stack of each thread in the module is to be adjusted according to the needs of that thread.

Some constants have to be adjusted when the onboard software is deployed. For instance the size of the queue in the Scheduler, the size of the queue in the Telemetry Manager, the allowed time frame in which the modules have to respond to an alive command, and the default logging period for polling housekeeping data.

8.3 New Features

There are a lot of extra features, which can be implemented in the system. An issue could be to make it possible to use dynamic size of packets, because of the large overhead, due to the fixed packet size.

As it is now the modules are initialised by a main program. This functionality could be given to the Platform Manager, which also could be responsible for starting and stopping the modules as needed.

The current housekeeping data generated by the software platform should be considered as preliminary and it is recommended that the group responsible for operating the satellite looks deeper into the subject.

9 Conclusion

The goal of this project was to create a satellite software infrastructure, which makes it possible to share the single onboard computer of the DTUsat between the different subsystems of the satellite. We have succeeded in creating the infrastructure although some things still are missing; the most important must be a structured test.

The concept of a module has been formulated in order to separate the software parts of the different subsystems, thus making concurrent development and testing easier. Separation between the modules was accomplished by letting a Packet Router handle all inter-module communication.

The chosen processor unfortunately does not have a memory management unit, so it is not possible to guarantee, that a module does not write in another module's memory space, thus the integrity of the entire software platform could be compromised by an error in a single module.

The time can cause trouble if the right procedures are not satisfied, so it is the Ground Segment's responsibility to ensure that this is done.

In order to reduce the complexity of the onboard software it was decided to let the Ground Segment handle errors. If an error occurs, a message is sent to the Ground Segment, in the form of a Telemetry packet - KISS was after all one of the design goals.

A Global Constants

In the software modules to the satellite, we have used a lot of predefined constants. The length of a sequence number is 2 bytes and this information is used in a lot of various files, but by using the define constructor in C we can use a symbolic name like SEQ_LENGTH instead of 2. This makes it very simple to change the value of a constant, hence making the system flexible to changes.
The file global.h includes all the system's constants.

A.1 Packet Constants

This section lists the most important constants and describes their use.

NAME	**Description**	**Value**
ID_LENGTH	The byte length of the source id	1
MSGTYPE_LENGTH	The byte length of the messages type	1
LENGTH_FIELD	The byte length of the field, which holds the length of the message data field	2
SEQ_LENGTH	The length of the sequence number	2
CMD_LENGTH	The length of the command	1
TIMESTAMP_LENGTH	The length of the timestamp	4
ERROR_LENGTH	The length of the error	1
PRIO_LENGTH	The length of the priority field	1
TELEDATA_DESC_LENGTH	The length of teledata description field	1
HEADER_LENGTH	ID_LENGTH + MSGTYPE_LENGTH + LENGTH_FIELD	
CMD_HEADER_LENGTH	SEQ_LENGTH + CMD_LENGTH	
CONF_HEADER_LENGTH	SEQ_LENGTH + TIMESTAMP_LENGTH + ERROR_LENGTH	
TELEDATA_HEADER_LENGTH	TIMESTAMP_LENGTH	
MAX_ARGSIZE	MAX_PACKETSIZE - CMD_HEADER_LENGTH	

continued on next page..

RESVALSIZE	MAX_PACKETSIZE - CONF_HEADER_LENGTH - ERROR_LENGTH	
MAX_TELEDATASIZE	MAX_PACKETSIZE - TELEDATA_HEADER_LENGTH	
PRIO_LOW	Low priority	0
PRIO_MIDDLE	Middle priority	1
PRIO_HIGH	High prioity	2
MSGTYPE_CMD	Command message type	0
MSGTYPE_CONF	Confirmation message type	1
MSGTYPE_TELEDATA	Telemetry data message type	2
PLM	Platform Mangager	1
SCH	Scheduler	2
HOC	Housekeeping Collector	3
TEL	Telemetry Manager	4
RAD	Radio	5
ATT	Attitude	6
POW	Power	7
CAM	Camera	8
TET	Tether	9
GND	Ground Segment	50
CMD_ALIVE	Alive request command	0
CMD_HKDATA	Housekeeping data request command	1
CONFIRM_ALIVE_ACK	Positive response to a CMD_ALIVE request. Meaning all threads are ok	0
CONFIRM_ALIVE_NACK	Negative response to a CMD_ALIVE request. Meaning some threads are dead	1
CONFIRM_HKDATA	Response to a CMD_HKDATA request	0

B Reserved Telecommand Numbers

10 command numbers are reserved for each module in the following way:

0-9 are reserved for global commands.

10-19 are reserved for the PLM module.

20-29 are reserved for the SCH module.

30-39 are reserved for the HOC module.

40-49 are reserved for the TEL module.

50-59 are reserved for the RAD module.

60-69 are reserved for the ATT module.

70-79 are reserved for the POW module.

80-89 are reserved for the CAM module.

90-99 are reserved for the TET module.

The command numbers 100-255 are free to use in coordination with the other module groups.

In the file global.h you can see, which command numbers within the reserved ones are already used by the modules constituting the basic software platform i.e. the modules PLM, SCH, HOC and TEL.

C DTUsat Library Functions

The following functions are offered by the DTUsat library.

Extracting information from packets:

- getFromID
- getMsgType
- getLength
- getTelecommand
- getArg
- getPriority
- getSeqNo
- getError
- getTimestamp
- getData

Creating packets:

- getNewSeqNo
- createCmdPacket
- createConfPacket
- createTeledataPacket

Sending and receiving packets:

- sendMsg
- sendMsgToEarth
- getMsg

Packet data manipulation facilities:

arraycopy

insertarray

extractarray

write2byteUnsignedInt

read2byteUnsignedInt

write4byteUnsignedInt

read4byteUnsignedInt

Time facility:

getCurrentTime

Error reporting facilities:

reportUnknownMsgType

reportUnknownConfType

reportUnknownCmdType

D Example

This is a skeleton module from which all modules are build. Whenever the term 'CAM' is used in the following it should be changed to the appropriate module when used for another module.
The function camp() is to write debug information and can be switched on and off in global.h. Each module have its own debug function made up of *prefix+p*.

```
//Used for checking if all threads are alive when CMD_ALIVE are received
#define CAM_NUMBEROFTHREADS 2

//Gives the module access to DTUSAT constants and communication functions.
#include "../system/global.h"

void cam_aliveresp();
void cam_housekeeping();
void cam_scheduler(cyg_addrword_t);

cyg_thread cam_thread_s[2]; /* Space for two thread objects */

//Space for each threads stack.
char cam_stack1[4096];
char cam_stack2[4096];

// Thread handles
cyg_handle_t cam_inbox_thread, cam_main_thread;

cyg_thread_entry_t cam_inbox, cam_main;

// list that holds a timestamp set by each thread in module
// Used in order to check whether all threads are alive.
cyg_tick_count_t thread_alive_list[CAM_NUMBEROFTHREADS];

//Mutex used to provid a protected area.
cyg_mutex_t thread_alive_list_lock;

//Space for the number of packets the module wishes to use.
//Care should be taken that different thread does not use the same packet
//at the same time.
char cam_inboxpacket[MAX_PACKETSIZE], cam_outpacket[MAX_PACKETSIZE];

int cam_init(void)
{
  cyg_mutex_init(&thread_alive_list_lock);

  //Create all the threads needed with belonging stacks.
  //More can be created as needed.
  cyg_thread_create(4,
                    cam_inbox,
                    (cyg_addrword_t) 0,
                    "cam_inbox_thread",
```

```
                    (void *) cam_stack1,
                    4096,
                    &cam_inbox_thread,
                    &cam_thread_s[0]);

  cyg_thread_create(4,
                    cam_main,
                    (cyg_addrword_t) 1,
                    "cam_main_thread",
                    (void *)cam_stack2,
                    4096,
                    &cam_main_thread,
                    &cam_thread_s[1]);
  //After creation of threads, start them.
  cyg_thread_resume(cam_inbox_thread);
  cyg_thread_resume(cam_main_thread);
  return 0;
}

// Thread number 0
// All thread should have a thread dedicated to reading from the
// inbox, as get() blocks.
// This function handles the minumum required message types for a
// module.
// Each module should evaluate what is to be done the places
// where printf are used.
// Properly the right thing is to send a message to Eart.
void cam_inbox(cyg_addrword_t cam_data) {
  char cam_inboxcmd;

  for(;;) {
    //This is the only place a module is allowed to read from its
    //inbox
    camp("cam waiting for message packet to be placed at add=%d.\n",
          cam_inboxpacket);
    getMsg(CAM, cam_inboxpacket);
    switch (getMsgType(cam_inboxpacket)) {
    case MSGTYPE_CMD :
      cam_inboxcmd = getTelecommand(cam_inboxpacket);
      camp("CAM recieved in cam_inbox: command=%d\n",   cam_inboxcmd);
      switch (cam_inboxcmd) {
      case CMD_ALIVE :
        cam_aliveresp();
        break;
      case CMD_HKDATA :
        cam_housekeeping();
        break;
      default   : //unexpected command
        printf(" * * * * * * * * * * * * * * * * * * * * * * * * * * *\n");
        printf(" * * * CAM unknown command type ....... %u     * * *\n",
                cam_inboxcmd);
        printf(" * * * SeqNo : %u     * * *\n", getSeqNo(cam_inboxpacket));
        printf(" * * * FromID: %u     * * *\n", getFromID(cam_inboxpacket));
        printf(" * * * * * * * * * * * * * * * * * * * * * * * * * * *\n");
```

```
        // report error to earth.....
        reportUnknownCmdType(CAM, cam_inboxpacket);
        break;
      }
      break;
    case MSGTYPE_CONF :
      switch (getError(cam_inboxpacket)) {
        //case..break;
      default   : //unexpected confirmation
        printf(" * * * * * * * * * * * * * * * * * * * * * * * * * *\n");
        printf(" * * * CAM unknown confirmation type .......   * * *\n");
        printf(" * * * * * * * * * * * * * * * * * * * * * * * * * *\n");

        // report error to earth.....
        reportUnknownConfType(CAM, cam_inboxpacket);
        break;
      }
      break;
    default : //unexpected message type
      printf(" * * * * * * * * * * * * * * * * * * * * * * * * * *\n");
      printf(" * * * CAM unknown message type .......        * * *\n");
      printf(" * * * * * * * * * * * * * * * * * * * * * * * * * *\n");

      // report error to earth.....
      reportUnknownMsgType(CAM, cam_inboxpacket);
      break;
    }
  }
}

// Thread number 1
// This thread is the one which should do the work.
// It is important that it regulary remember to record that it
// is alive,
// so only timed waits must be used
void cam_main(cyg_addrword_t cam_data) {

  for (;;) {

    // record that this thread is alive (must be done regulary)
  cyg_mutex_lock(&thread_alive_list_lock);
    thread_alive_list[1] = cyg_current_time(); // Thread no 1
  cyg_mutex_unlock(&thread_alive_list_lock);

    //do something
    cyg_thread_delay(2000);
    //set time stamp
  }

}
// When a CMD_ALIVE is received this function replies.
// If any other threads have not update their timestamp an NACK is
// send.
```

```
// Othervice ACK is send.
void cam_aliveresp() {
  int i;
  cyg_tick_count_t ct, oldest_ack, maxdiff;

  ct = cyg_current_time();

  // The inbox thread (0) must be alive now... (this thread)
  thread_alive_list[0] = ct;
  oldest_ack = ct;

  //Check time stamp from other threads in this module

  // find oldest timestamp
  cyg_mutex_lock(&thread_alive_list_lock);
  for(i=0;i<CAM_NUMBEROFTHREADS;i++)
    if (thread_alive_list[i]<oldest_ack)
      oldest_ack = thread_alive_list[i];
  cyg_mutex_unlock(&thread_alive_list_lock);

  maxdiff = ct - oldest_ack;

  //if ok then create ACK confirmation
  if(maxdiff <= THREADMAXTIMESINCEACK){
    createConfPacket(CAM,         // Source (this module)
        0,                        // Length of extra values returned
        getSeqNo(cam_inboxpacket),// Sequence number (taken from cmd packet)
        CONFIRM_ALIVE_ACK,        // Errorvalue (CONFIRM_ALIVE_NACK if something is wron
        0,                        // Pointer to extra values
        cam_outpacket);           // Where to place the created stuff
    plmp("CAM: send Alive confirmation\n");
  }
  else{
    createConfPacket(CAM,         // Source (this module)
        0,                        // Length of extra values returned
        getSeqNo(cam_inboxpacket),// Sequence number (taken from cmd packet)
        CONFIRM_ALIVE_NACK,       // Errorvalue (CONFIRM_ALIVE_NACK if something is wron
        0,                        // Pointer to extra values
        cam_outpacket);           // Where to place the created stuff
    plmp("CAM: send ***NACK*** Alive confirmation\n");
  }
  /*send packet*/
  sendMsg(PLM, cam_outpacket);
}

// This fuction is called when a CMD_HKDATA is received.
// It should send a packet to Earth containg the house keeping data.
void cam_housekeeping() {
        /*create and send teledata packet with housekeeping data*/
  //...
  /*create and send confirmation to HOC*/
  //When the House keeping data is send an confirmation should be sent.
  createConfPacket(CAM, 0, getSeqNo(cam_inboxpacket),
                   CONFIRM_HKDATA, 0, cam_outpacket);
```

```
  sendMsg(HOC, cam_outpacket);
  camp("CAM: have just send     hkdata confirmation\n");
}
```