Logging system for NTNU test satellite

Erik Hals

December 19, 2014

Preface

This report is my submission for the course *TFE4520/ TFE4521- Design av digitale systemer, fordypningsprosjekt* 7,5sp.

In this project, the task was to design the logging system for NUTS.

I wish to thank Roger Birkeland, Amund Gjersvik and Bjørn B. Larsen for information, help and guidance throughout the project work.

Problem Description

As a part of the national student satellite program, the NUTS CubeSat project was initiated at NTNU in 2010. The projects goal is to give hands-on experience to students within different fields of satellite technology. This includes planning, specification, design, construction, launch and operation of a satellite.

The main parts of the satellite bus are:

- Power system (solar cells and batteries)
- Attitude and orbit control
- Mechanical system (structures and mechanisms)
- TT&C (telemetry, tracking and command)
- Thermal system
- On-board data handling
- Payload and experiment

The NUTS CubeSat is using the AVR32 UC3 MCU as main processor. There are two processors on board, one for the main computer and one for the communications system. The system is running the FreeRTOS operating system.

In this assignment, the student should design the logging system for NUTS. The framework must be outlined and planned. The system must write (and extract) different parameters and events to a file system. An event log must be made, in addition to statistical logs to hold for example voltages and current consumption over time, the number of errors (and their types), restore logs (register values and such). The logger must run as a FreeRTOS task.

The student is required to joins weekly meetings with the rest of the team (consisting of staff, other project/master students and other students) to share experience and knowledge.

Contents

Chapter 1

Project Introduction

> The NUTS student satellite project is organized and supported by Department for Electronics and Telecommunications (IET). Our aim is to design, build, test and launch a double CubeSat by 2014. Most of the work is carried out by students from several departments and study programs, such as electronics, communications, space technology, physics, cybernetics, computer science, mechanics and more.
>
> Our project is part of the The Norwegian Student Satellite Program at NAROM (Norwegian Centre for Space-related Education), together with CUBEstar at University of Oslo and HINCube at Narvik University College.[3]
>
> -Retrieved from the NUTS website, about the project.

This report is written under the assumption that the reader is familiar with the NTNU test satellite. For more general information about the project, and the completed work, it is recommended to read the NUTS mission statement[4] and look at the information on project website[5].

1.1 Scope

This project aims at designing the logging system. The main challenges will be to create a logging system that can work with the already implemented parts of the project. Many factors is already decided, such as communication between modules, protocols, operating system, memory, and microcontroller devices. A

significant part of this project is devoted to becoming more familiar with previous work as well as reviewing the literature.

The project will mainly focus on designing the framework, file format, and communication with the system. There are however parts of the design work of the system that are not looked at in this project, before the system can be implemented. These are file system for the flash memory, and how to design software in order to make it more resistant to single event upsets. The reason for this is because these are assignments that are general to the whole satellite, not only the logging system, and are separate tasks other members of the NUTS team are working on.

1.2 Previous work

Most of the work has been radio and hardware design related, and very little software for the OBC is finished. There are however an experts in team report from 2013 were a logging system has been designed and implemented[1]. This logging system polled sensors and stored their values as an event log on the ram. This system however is created only for this task, and is not modular in any way and did not take into account CSP, multitasking or flash implementation. The work gave some insight to the satellite and communication between sensors and the OBC, but no parts of the system could be further developed into a logging system for the satellite.

Chapter 2

Logging system introduction

The logging system plays an important role in debugging and error correction of the satellite. An error may be intermittent and hard to track down, but a logging system can aid the process of investigating issues significantly. The system has the job of collecting data from other modules and applications, and store them for later analysis. Additionally is has to do this in an efficient and correct manner. A log with errors can be worthless and if the logging system is too demanding on the system and erroneous, it defeats its purpose. One therefore have to carefully look at what data needs to be logged, and how this should be done.

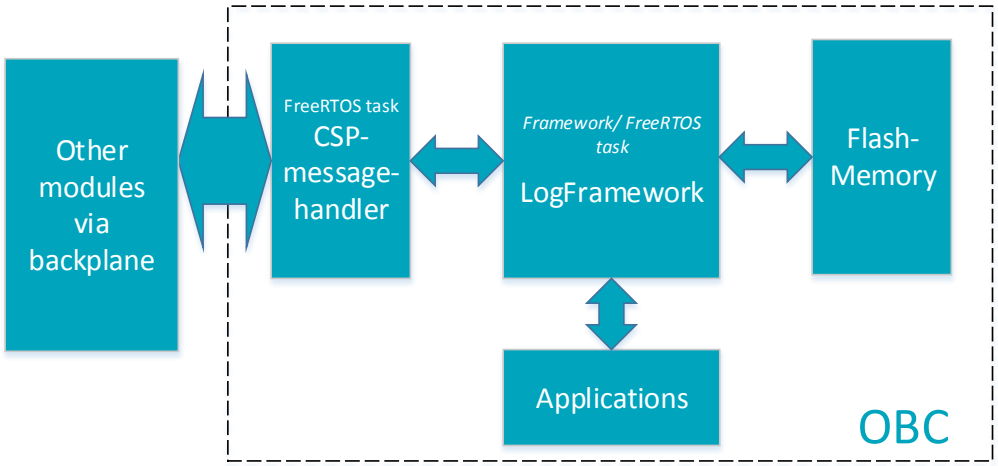

Figure 2.1: General overview of the system modules

2.1 Requirements

The system has to be be designed for two different scenarios. During development, the system will be used for debugging purposes, it should give the user access to as much data as possible, and be able go gather this.
When the satellite is in flight, the system has to work autonomously, gather only crucial data, and transmit this to the ground. It must detect erroneous conditions, and act on it.
The systems is required to run as a FreeRTOS task. With this one has to consider, topics such as scheduling, and being able to be preempted. Additionally memory has to be allocated for data yet to be logged.
In order to control the system, a command set, and functions has to be specified for other modules and applications to use. Modules such as radio, ADCS, and payload will communicate with the OBC via CSP on the I^2C bus. While an interface has to be made for the applications running internally on the OBC.
A filesystem is needed in order to provide a more secure way of storing data on the flash memory. It also need to keep track of wear leveling.

2.2 Specification

In order to make changes to a part, or a functionality of the system, the logging system is designed in a modular manner. This is done to ease further development or changes to the system. With a large project such as the NTNU test satellite, new ideas and improvements are made each semester, and it may require big changes in the existing work, and modules may be rendered obsolete. If parts of the system can be further developed, rather than redesigned, significant amounts of work can be saved. Based on the requirements, the system is broken down, and specified in the following manner.

2.2.1 Framework

The main framework of the logging system log.h, will be included and run as a FreeRTOS task. It will mainly have the responsibility of coordinate every action in the system, making sure modules can send it commands with data to be logged. It will also have to know where logs can be found, in order to retrieve these for transmission to the ground. If an error or abnormal behavior should occur in the system, this module has the responsibility of acting upon it, restore functionality, and notify the ground station that an error has occurred, or have the option of

being reconfigures from the ground. Log.h will interface with other applications running on the OBC such as the CSP handler and sensor logger, and the flash memory.

2.2.2 CSP Handler

The CSP handler is the main communication interface between other modules and the logging system. It has to take care of the hardware- and link layer workings of the I^2C bus, as well as the network layer workings of the CSP protocol. This handler should run as a separate FreeRTOS task with the ability of being interrupt triggered, or preempted in another manner. It also need to have a buffer for storing incoming packages waiting to be processed by the main logging task. The CSP handler should not only handle messages for the logging system, but for the entire OBC.

2.2.3 Sensor Logger

In order to log the values from the multitude of sensors, they need to be polled. A task will be charged with this, and log data within an interval of a few seconds, and then store the values using the logging system.

2.2.4 File System

The main task of a file system is to provide the format on how data should be stored, as well as wear leveling and block managemet. The file system will also have to suit the already implemented flash memory.

Chapter 3

Logging System Design

3.1 Logformat

In order for the system to interpret and make sense of what's being logged, a format has to be defined. This can be done in a number of ways, but it is important that it is done in a way that reduces workload in the CPU and the likelihood of errors. If there is an error in the log, induced by either bit flip, bug, or other events, the format should be simple enough for errors to be picked up, either by the OBC itself or humans when the log is downloaded and viewed, and should at least not cause propagating errors on the OBC, due to errors in the log file. For this not to happen there has to be security mechanisms in the function that has the task of creating, writing to, and reading from logs. As well as the file system, that organizes the data, and handles the low level workings of the flash memory. Syslog is a standard for logging in computers, and it is used for computer system management and debugging[2]. Syslog specifies the message format, with codes for facility (e.g. kernel messages, user-level or log alert) and severity (e.g. emergency, warning and debug level). It is however created for computer systems, and not embedded hardware. Many of the functions are obsolete for our application, and will create more overhead than necessary. Also there is not going to be exchanged logging information between modules, only modules to log to ground, a standard the message format is therefore not as strict.

3.1.1 File format

The log file itself should be saved as a comma separated values (CSV) file[6], this is a well known method of saving data, although not a standard, it has been used

for a long time. The basic idea behind CSV is that each value should be separated with a comma in a line, and that each line is ended with line break(CRLF). This makes the log format very dynamic in regards to the number of parameters, and what format the parameters should have. For instance it does not matter if the parameters is a string, integer, symbol, or a combination of either. As long as they are separated by a comma it does not matter what is inside the parameter. It is also indifferent how many parameters is written in an entry, as long as it is ended with CRLF. Another entry can then be written on a new line.

In figure 3.1 we see an example on how a raw data of a CSV file is stored on the memory, and viewed in a program such as Excel. This is an example on how voltages and current from the INA219 sensor could be logged. The commas and CRLF will be stored as their corresponding ASCII values in the file, but are translated in this example for viewing purposes.

time,v1,v2,c1,c2CRLF
19:45:00,3.31,3.28,112,550CRLF
19:45:05,3.31,3.29,115,543CRLF
19:45:10,3.30,3.31,116,540CRLF

time	v1	v2	c1	c2
19:45:00	3.31	3.28	112	550
19:45:05	3.31	3.29	115	543
19:45:10	3.30	3.31	116	540

Figure 3.1: Top: raw CSV file. Bottom: Corresponding file opened in Excel

3.1.2 Event log

In order to meet different functionality needs, you need different logs. The first one is the regular event log. Here, new data is appended to the end of the previous, to form a list. There should however be a limit to how many entries the log can contain. With limited capacity on the data transmission to the ground, old data would simply take up space in the memory, and never be used.

3.1.3 Statistical log

The statistical log should calculate and store different statistical values of the data being fed to it. These values could be for instance average over different inter-

vals, minimum and maximum values, or weighted average. For calculating average when a new value are added to the set this formula can be used:

$$average_{new} = average_{old} + \frac{value_{new} - average_{old}}{size_{new}}$$

The statistical log will make it easier to transmit data to the ground, as it summarizes the data before transmission. Less data will need to be sent in order to provide the same information. For instance, having a statistical log gathering data about the battery voltage can immediately reveal if there was a power loss during the eclipse, and that the power consumption of the satellite needs to be reduced.

3.1.4 Restore log

Restore logs can perform permanent and safer storage of values. Registers, or other values you want to keep can be stored to the log, should something happen to the program running a task, and the processor has to be rebooted, you can load the values from the log, and continue execution. The restore logs will only contain one set of data, new values will not be appended to a list like event log, but overwrite the previous values.

3.2 Modules

The system consist of several modules. These are library files which are to be included either as FreeRTOS tasks, or part of them. The logging system as a whole are made in this modular manner because several parts of the final system is not yet completed, and how the logging system interfaces with the rest of the satellite is yet to be clearly defined. For instance there is yet to be implemented a file system for the flash memory, it would therefore be pointless to implement this in the main logging task, but rather design an interface module where the two parts are linked together. When a filesystem gets implemented you will only need to make changes to the interface functions, and not the main logging module. The same goes for the other end of the system, the one where other parts of the satellite interfaces with the logging system. If the radio for instance, wants to retrieve a log, in order to send it back the the ground, you need a communications handler to carry out the communication, between the modules.

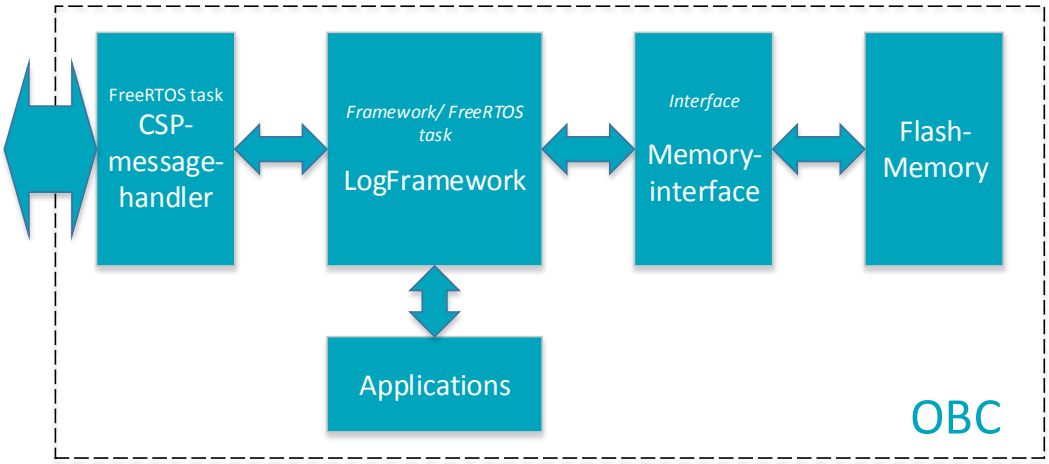

Figure 3.2: Overview of the logging system

3.2.1 Log framework

This is the main module for the logging system. It contains all the functions for writing, reading, and controlling the log system. All communication between other modules or tasks, and the logs goes through this unit.

Communication between the CSPhandler, applications and the framework can be done in several ways. But the primary form of intertask communications in

FreeRTOS are Queues. [7] This combined with binary semaphores can provide a way to exchange the data, as well as providing a blocking mechanism for when other tasks are communicating with the log framework, or it is busy.[8]

The log framework module should consist of a state machine for different modes of operation. In figure 3.3 we see the 3 states of operation. Idle for when the system is waiting for a packet. When a packet has been put into the queue and it is notified, the framework task will fetch the data from the queue, and look at the command set header, to get information on which command should be executed. (The command set is elaborated in section 3.3). If the packet contains data to be logged, it is then stored on the flash. The logging system should keep a file on the memory with configuration for the logging system. This file should contain information about the existing logs in the system.

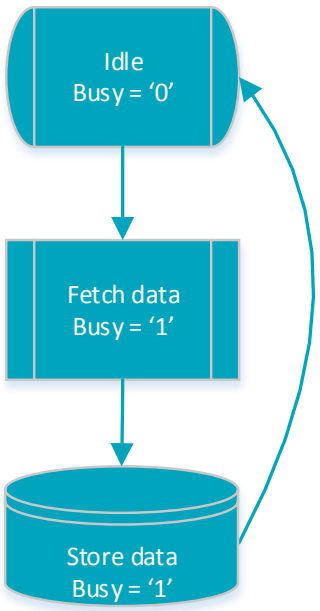

Figure 3.3: States of the log framework

3.2.2 CSP message handler

The message handler has the task of organizing communication between the OBC and the other modules via the backplane. It has to separate CSP packets from other I^2C communication on the bus. The handler needs to be preempted when a packet arrives on the bus, and have buffers to store the packets before forwarding them to the appropriate tasks. It should communicate with other tasks with Queues the same way as with the logging system.

In figure 3.4 we see how the the CSP header consists of various fields. Source and Destination, is used to tell who sent the packed, and where it is going. Each module in the satellite need to get their own address, in order for the different message handlers to know if they should listen to a packet or not. The next fields are the port numbers. This can be used to separate traffic between processes running on the different microcontrollers. Each process should get their own unique port numbers. This allows separation between packets for the logging system from packets to other applications.

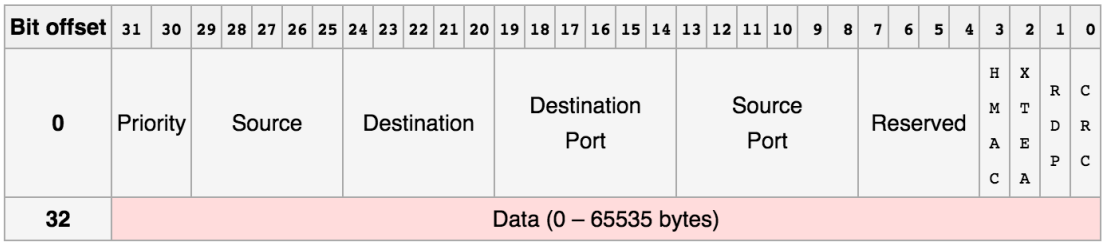

Figure 3.4: CSP header

3.2.3 Memory interface

Memory interface should be made as a header file to be included in tasks that perform actions on the flash memory. It has to contain the file system, and the low level workings of the flash. As more than one application will be able to access the memory, primitives need to be implemented in order to prevent multiple tasks trying to access the memory at the same time, for instance a binary semaphore.

3.2.4 Sensor logger

In order to log different sensors within the satellite one need an application to poll the sensors and get their value. This task communicates with different sensors on the I^2C bus, and uses the logging system to write to it's own log. This log will be used as a general satellite status/health log. The task will log at a certain interval, for instance every 5sec. An RTC should be added to the satellite for the log interval to be constant, and to timestamp the log. The task should contain a state machine similar to figure 3.3, where it is idle until a log interval timer runs out, then gathers all the sensor data, and finally stores it on the memory via the framework.

3.3 Command Set

The command set defines the language for communications with the logging system. There are three different types of commands that need to be specified. Write-, read- and control commands. The control commands should only be used to configure the system from the ground when the satellite is in flight. The commands will be contained within the data section of the CSP packets and will have the format as shown in figure 3.5.

3.3.1 Commands

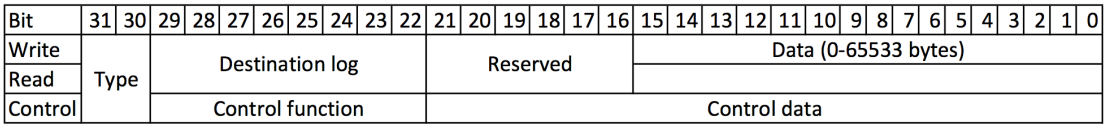

Figure 3.5: Command set header

For writing-, reading- or control commands, the type field should be set as follows:
Write commands: Type = ”00”
Read commands: Type = ”01”
Control commands: Type = ”11”

The ”Destination log” field refers to the identifiers for the logs. With 8-bit, you can have a maximum of 256 unique logs, tho this can be extended by the ”Reserved” field which is currently unused. The data field should contain the data one want to be logged. Due to the limitation of the CSP protocol only allowing 65535bytes, there is a limitation to how much data can be sent at one time. But 65533 bytes should be more than enough. For reading logs, for instance when the ground station requests a log, the radio module should be issued with sending a packed containing the correct type code (”01”), as well as the ”Destination log” number. The logging system will then return the data contained in the corresponding log file.
When the satellite is in orbit, it is essential to be able to control the logging system in the event of corrupt files, or if the system has failed in other way. One needs to be able to restore the functionality. The OBC can be rebooted, but the flash memory can not be restored. As a consequence commands are needed, in order to restore the structure of logs. In figure 3.5 we see that control commands corresponds to type = ”11”, ”Control function” corresponds to specific control

commands, (for examples see below) while "Control data" is information needed by the "Control function". In figure 3.6 we see functions that is needed to control the logging system.

Bit	31	30	29	28	27	26	25	24	23	22	21-0
Control			Control function								Control data
Create event log	1	1	0x01								Log number(0-255)
Create statistical log	1	1	0x02								
Create restore log	1	1	0x03								
Delete log	1	1	0x04								
Reset framework file	1	1	0x05								0

Figure 3.6: Control commands

3.3.2 Return values

Write commands

The return values for write commands should be pretty simple. There is only need for a confirmation that the data was successfully stored on the memory. For this, a byte containing only ones can be used. Different error codes can be implemented in bytes 0x00 to 0xFE, should a write command not be successful.

Read commands

This command should return the data in the requested log. If this log is not found, or any other error should occur, it should return 0x00.

Control commands

Control commands will require a wider variety of return values. For the create log commands, the log identifier needs to be returned, in order for the module or application to know the log identifier to be used in the write function. With the delete log, and reset framework file function a success or failure value should be returned. Here 0xFF and 0x00 can be used.

Chapter 4

Discussion

What should be logged by the Sensor logger?

The sensor logger should log voltages and current from the INA219 sensors. They should be assigned a statistical log to minimize the amount of data, and provide averages for 1 hour up to 1 week scope. Though temperature sensors were removed from the current versions of the modules, it could be useful to know the temperatures of different components should they fail. Also an RTC should be implemented, to give an accurate time, not only for the sensor logger, but all the logs.

What other data should be logged by the logging system?

The logging system is a service people can choose to use. It is up the the designers of the other modules to implement logging functions in their module or process. It is then my task to advertise the logging system, and show its use. All the modules in the satellite should use the logging system to log status messages. The EPS module could keep a statistical log of the battery voltage. The ADCS module could log how much it has to compensate in order to keep the wanted attitude. The radio could keep a log of the traffic, and the OBC could keep restore logs, for its register values. Also an event log should be used to keep track of exceptions such as an applications has failed.

How demanding will the system be on the bus and OBC?

This is hard to estimate, because it all depends on how much the logging system is going to be used. The microcontroller on the OBC is very powerful and should not have any trouble performing the calculations required. The bus however got limited capacity, and reading large files such as an event log could lead to capacity

issues. This have to be looked at after implementation to find out what the final footprint will be. The size of the logs can be reduced as a solution.

How can the logging system help debug the satellite in the lab before launch?

The logging framework could be set up with an UART connection. After receiving packages from the CSPhandler, or directly from an application it can direct the data to the UART port instead of saving to on the flash memory.

Chapter 5

Further Work

There are still some design work to be done. A filesystem will have to be sourced, and the memory interface will have to be designed. The system will then have to be implemented on the OBC, tested and verified. There are still uncertainties about the CSP protocol which affects the work done with the CSPhandler.

There is still some organizing work to be done with the logging system. Modules and applications that is going to implement a log, needs to be assigned a static log number for use with the logging system. Also port numbers will have to be assigned for the logging system as a whole.

The command set is very basic at the moment, more functions will probably need to be implemented in the future. It is hard to identify all the functions that will be needed, this will however be more apparent as the system gets implemented, and the true extent of the system is revealed.

Chapter 6

Conclusion

In this project the framework, CSPhandler and sensor logger was designed. Also the command set and file format was outlined and planned. Parts of the logging system such as the framework, and the sensor logger is ready to be implemented, while the flash memory interface needs a file system. The CSPhandler is still waiting for the decision on whether to use CSP or not, to find its use.

The main challenges in this project was to get familiar with the work that has already been done. There are many reports and information to find, but a lot is obsolete, and it was hard to keep track of what is current or not.

Bibliography

[1] Systemovervkning i studentsatellit; Jonassen, Mikkelsen,Teilgrd, Edvardsen, Arnesen, Andersen

[2] http://en.wikipedia.org/wiki/Syslog

[3] http://nuts.cubesat.no/about-the-project

[4] http://nuts.cubesat.no/upload/2012/01/20/nuts-1_mission.pdf

[5] http://nuts.cubesat.no/

[6] http://www.ietf.org/rfc/rfc4180.txt

[7] http://www.freertos.org/Embedded-RTOS-Queues.html

[8] http://www.freertos.org/Embedded-RTOS-Binary-Semaphores.html