

TFE4521 – Specialization Project

---

# Implementing CSP over I<sup>2</sup>C for the new repository on the NTNU Test Satellite

---

*Author:*  
Erlend Riis Jahren

*Supervisor:*  
Prof. Bjørn B. Larsen



Department of Electronics and Telecommunications  
Norwegian University of Science and Technology  
Norway  
December 19<sup>th</sup> 2014

### **Abstract**

The following report presents the work and results of a project done in association with the NTNU satellite project, NUTS. The main assignment was to implement CSP, the CubeSat Space Protocol, as a network communication layer in the software of the satellite, thus simplifying the communication link and facilitate further development. The implementation itself did not include the software development of the protocol, but rather an installment of a published library from GomSpace. Also, source code supporting I<sup>2</sup>C was added to the software repository, and a controller for sending messages over CSP was implemented. The implementation was then tested for its basic functionality, and the results were discussed. Finally, a short discussion about the structure of how an application layer can be implemented is presented for further development.

# Preface

This report documents the implementation and testing process of a network communication layer on the NTNU Test Satellite performed during the fall of 2014. The project aimed at developing the satellite towards final specification, and will hopefully be of help to further develop the internal communication structure.

Special thanks goes to Bjørn B. Larsen and Roger Birkeland for guidance and help during the course of the project. Other contributions by fellow students have also been greatly appreciated.

# Contents

<b>Preface</b>	<b>i</b>
<b>1 Introduction</b>	
1.1 Project description . . . . .	1
1.2 Out of scope . . . . .	2
<b>2 Context</b>	
2.1 CubeSat . . . . .	3
2.2 CubeSat Space Protocol (CSP) . . . . .	4
2.3 I <sup>2</sup> C . . . . .	4
2.4 FreeRTOS . . . . .	7
2.5 Glue layer between CSP and I <sup>2</sup> C . . . . .	7
<b>3 Implementation</b>	
3.1 Installing the CSP library . . . . .	9
3.2 Importing I <sup>2</sup> C drivers . . . . .	10
3.3 Importing glue . . . . .	10
3.4 Simplified CSP API . . . . .	11
3.5 CLI control . . . . .	12
3.6 Example code for testing . . . . .	14
<b>4 Testing</b>	
4.1 Test Setup . . . . .	17
4.2 Tests . . . . .	18
<b>5 Results</b>	
5.1 Using CLI to test . . . . .	21
5.2 Test results . . . . .	21
5.3 Issues . . . . .	23
<b>6 Discussion</b>	

6.1	Application layer protocol options . . . . .	27
6.2	I <sup>2</sup> C without CSP . . . . .	28
<b>7</b>	<b>Conclusion</b>	<b>29</b>
	<b>References</b>	<b>30</b>
	<b>List of Figures</b>	<b>32</b>
<b>A</b>	<b>Step-by-step guide to installing CSP</b>	<b>33</b>
<b>B</b>	<b>How to use CSP</b>	<b>38</b>
<b>C</b>	<b>Source code</b>	<b>42</b>
C.1	CSP CLI interface . . . . .	42
C.2	CSP example code . . . . .	44
C.3	CSP controller code . . . . .	49

# Chapter 1

## Introduction

*“Space isn’t remote at all. It’s only an hour’s drive away, if your car could go straight upwards.”*

– Sir Fred Hoyle, *September 1979*.

The NTNU Test Satellite, abbreviated NUTS, is a Norwegian CubeSat project aimed at successfully deploying a satellite made mainly by students from the Norwegian University of Technology and Science. The project is a successor of NTNU’s nCube-1 and nCube-2 projects which both failed to become operational due to external launch and deployment issues.[16] The satellite, which is expected launched within 2016 or 2017, have lately been revised to assure correct behavior and fault tolerance, thus requiring some parts to be reimplemented.

### 1.1 Project description

The goal of this project is to implement a network layer protocol for communication between modules on the NTNU Satellite. From previous work within the NUTS project, it has been decided that the CubeSat Space Protocol(CSP) will be an appropriate protocol, as it provides advanced functionality while simplifying further communication development and implementation.

In addition, the project will cover a short overview of format options for the

payload packets, such as fixed/variable messages or splitting/joining of data, and propose an overall solution for the communication structure.

This report covers the implementation and testing of CSP with I<sup>2</sup>C support running on FreeRTOS.

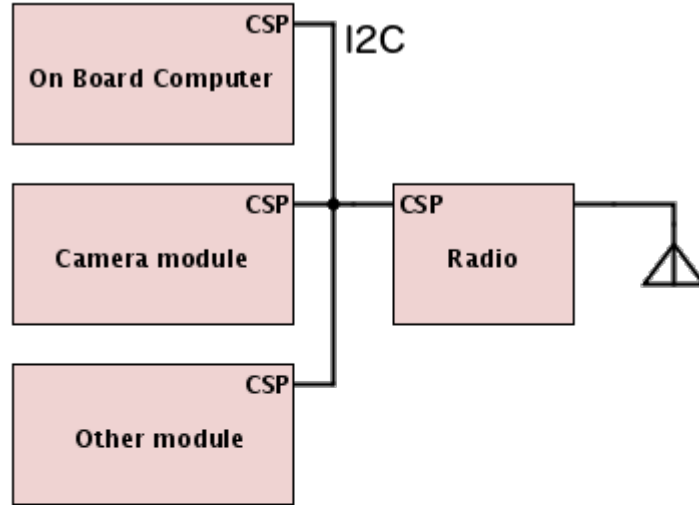


Figure 1.1: Implementing CSP over I<sup>2</sup>C for the NUTS satellite

## 1.2 Out of scope

Space environments is known to introduce unexpected errors on electronic systems due to radiation passing through the circuits.[14, p. 5] Although this is an important factor when designing and implementing software for the satellite, it will be considered out of scope for this project. This is mainly because no specific fault tolerant implementation techniques have been decided upon by the NUTS project group.

## Chapter 2

# Context

In 2012, Andreas Giskeødegård implemented a version of the CSP library for the software repository NUTS used at that time.[4]. His report, *Implementing CSP over I2C on the NTNU Test Satellite*, described the CSP implementation process, its interaction with subsystems, and discussed further improvements. The implementation was later properly tested by a group of students from the NTNU subject "Experts in teams"(EiT), where it was tested for various possible shortcomings and situations that may arise in the satellite.[1] Giskeødegård's implementation of CSP was proved to be successful, yielding high transmission rates with low probability of erroneous transmissions. However, the tests yielded certain issues regarding error detection. The issues regarded faulty return values in the CSP library methods, more precisely not receiving indicators of errors when transmissions were unsuccessful[1, p. 29] The source of these bugs were never found.

### 2.1 CubeSat

A CubeSat is a small satellite built within standardized dimension units(U) of 10x10x10 cm, and a weight less than 1.33 kg per U. The original standard only defines CubeSats up to 3U. (30x10x10 cm)

The standardization of the shape, size and weight significantly simplifies the launching mechanisms by using a common deployment system, which again significantly reduces the cost of getting the satellite into orbit. This makes



the CubeSat a popular specification among non commercial organizations and universities.[11]

## 2.2 CubeSat Space Protocol (CSP)

In 2008, students from Aalborg University in Denmark began the development of what is now known as the CubeSat Space Protocol. Their work resulted in a small network/transport layer protocol specifically designed for CubeSats.[6] The implementation is written in GNU C, and is currently runnable on FreeRTOS, POSIX systems, MacOS and Windows. Originally, the MAC-layer driver was written to support CAN-bus, but this has later been extended to include I<sup>2</sup>C , Spacelink and RS232 interfaces.[8] The source code of the implementation is released under a LGPL licence, which permits copying and redistribution of verbatim copies, while applying changes to the original, or, distributing a modified copy under the same name, is not allowed.[5]

Offset	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	Priority	Source						Destination				Destination port				Source port				Reserved	HMAC	XTEA	RDP	CRC								
32		Data (0-65536)																														

Figure 2.1: CSP header structure

CSP enables multiple submodules of an embedded system to communicate through a service oriented network architecture.[7]. This allows developers to use the communication bus as interface between subsystems, hence only needing to decide on a service contract with ports that each subsystem will respond to.

## 2.3 I<sup>2</sup>C

Communication between modules on the NUTS satellite happens over an I<sup>2</sup>C bus, using the satellites backplane as the physical link as illustrated in Figure 2.2. I<sup>2</sup>C is a well known and popular protocol for serial communication over a two-wire interface(TWI).[10] The protocol was designed by Philips to simplify communication between interconnected components.

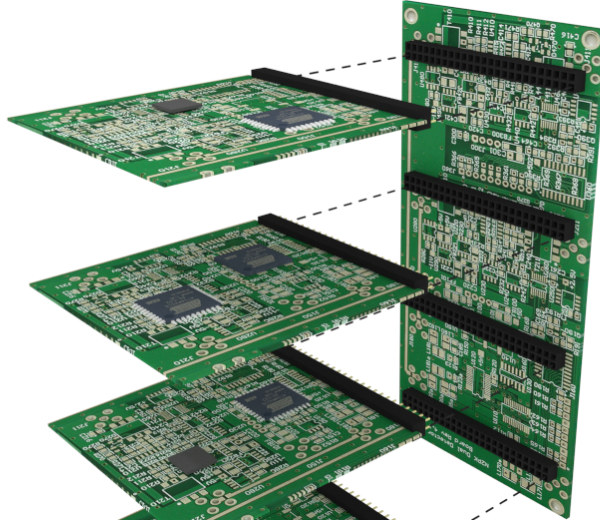


Figure 2.2: The illustration shows how all modules on the satellite is connected through a backplane.

The protocol is generally designed for slower communication links, with an original speed limit of 100kbit/s. This has later been redesigned to allow for faster transmissions with the fastmode of 400kbit/s, the high speed mode yielding 3.2Mbit/s and in 2012, an ultra fast mode was released, with transmit data rates of up to 5Mbit/s.[9][12] Still, I<sup>2</sup>C has other beneficial features, which are probably the reasons for making it so popular:

- A simple setup consisting only of two signal lines
- Basic Master/Slave connections between components in the bus
- The protocol supports multiple masters within the same bus, offering arbitration schemes, address decoding and collision detection
- Baudrates are specified by the masters, and are not bounded by preset values
- Addresses are implemented through software

### 2.3.1 Physical structure

The signal lines on the I<sup>2</sup>C bus consists of a SCL - *clock signal*, and a SDA - *data signal*. Both of the signals are bidirectional, where the SCL is initiated by the master when transmitting, and the SDA signal is used by both master and slave during transmission. The signals are designed as open-drained, with pullup-resistors on both lines to a common voltage source.

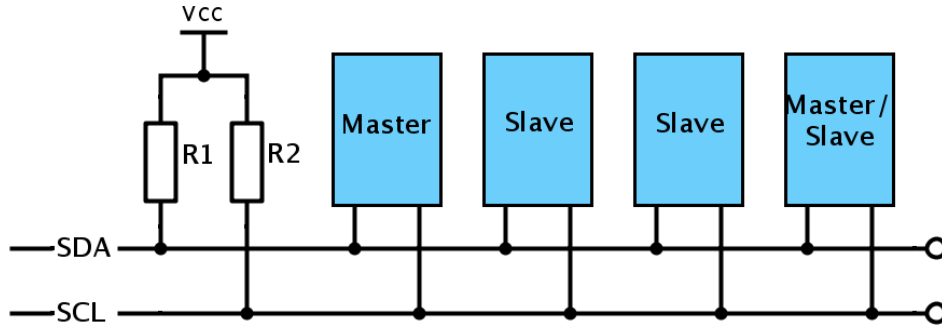


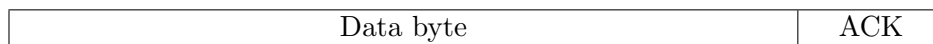
Figure 2.3: The physical structure of an I<sup>2</sup>C link.

### 2.3.2 Transmission phases

A successful transmission consists of several events on the SDA signal line.[13, p. 9-10]

**START** Initiation of a transmission is accomplished by making the master pull the SDA signal from HIGH to LOW while the SCL signal is HIGH, as illustrated in Figure 2.4. When this condition happens, slaves on the bus recognize that a transmission has started, and will listen to the signal to check if the message is addressed to them.

**Sending** Transmissions consists of a byte followed by an acknowledgement bit. There is no restrictions on the number of sequential bytes to be sent in a transmission.



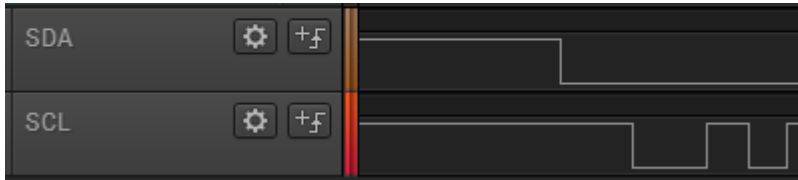


Figure 2.4: Initiation of a transmission

**Acknowledgements** After each transmitted byte, the receiver module of the transmission must pull the SDA signal LOW for one bit to indicate that the byte has been received. If the SDA is not pulled low during the 9th bit, it will indicate a Not Acknowledge signal. This results in a termination of the transmission.

**STOP** The termination of a transmission is accomplished by transitioning the SDA signal from LOW to HIGH while keeping the SCL HIGH.

### 2.3.3 Arbitration

Each master that wants to transmit on the I<sup>2</sup>C -bus listens to the SDA signal to see if other masters are trying to access the bus simultaneously. If another master has pulled the signal low, the first master will not be able to start its transmission.

## 2.4 FreeRTOS

The OS used on the satellite, called FreeRTOS, is a market leading, cross platform, real time OS that is widely used for micro-controllers and microprocessors.[3] FreeRTOS provides parallelism in submodules of the satellite through its use of prioritized tasks, while also supporting numerous features including memory management, semaphores, timers and more.

## 2.5 Glue layer between CSP and I<sup>2</sup>C

In 2011 CSP was partly commercialized as it was handed over to GomSpace, a Danish company providing out-of-the-box solutions for nano- and pico

satellites.[6] As a result of this, the code for the layer between CSP and I<sup>2</sup>C is not freely available.

This issue was solved when Andreas Giskeødegård designed and implemented the CSP-to-I<sup>2</sup>C glue layer and the adjoining drivers for the NUTS project in 2012.[4, p. 19] In short, the glue works as the connector of the CSP library and the I<sup>2</sup>C source code. The CSP library calls a set of preset methods that are referenced to the glue, which forwards the method operation to I<sup>2</sup>C . The glue also connects the two protocols the other direction, namely that the I<sup>2</sup>C handler will call methods that forward the information to the CSP library.

## Chapter 3

# Implementation

Giskeødegård designed and implemented CSP, an I<sup>2</sup>C driver, and the CSP-to-I<sup>2</sup>C glue for the old NUTS repository in 2012[4]. The implementations in this project will take use of Giskeødegård's work and make it function properly in the new repository. To separate between previous and current implementations of the CSP system, they will from here be referred to as new and old.

### 3.1 Installing the CSP library

The CSP library is download-able from GomSpace's page on [github.com](https://github.com). The file contains all source code and a script that simplifies the build and install process. The script, called *wscript* is run with *waf*, a python based general-purpose build system made available as open source.[15]

For a correct build of the library, compiler flags specifying the board, part and architecture that the library was aimed for needed to be added in the *wscript*. Further configurations were set as parameters when running the script, including specified compiler, link layer protocol(I<sup>2</sup>C ), OS, and a reference to certain header files in the repository it is compiled for.

In its entirety, the *waf* install process is quite unintuitive. To help other NUTS students in future projects where a CSP install is necessary, a step-

by-step guide was created. The guide is available on the NUTS wiki page<sup>1</sup>, and Chapter Step-by-step guide to installing CSP in the appendix of this report.

## 3.2 Importing I<sup>2</sup>C drivers

Giskeødegårds report *Implementing CSP over I2C on NTNU Test Satellite* presents an implementation of a dual master/slave driver for I<sup>2</sup>C communication, introducing a tri-state state machine to control the actions of modules connected to the bus.[4, p. 16-18]. The state machine listens to the bus in a IDLE state, and transfers into a SLAVE state when it recognizes its I<sup>2</sup>C address on the bus. After the transmission is completed it resets back to its IDLE state. Whenever the module needs to initiate a communication, it moves into a MASTER state, enabling control of the bus.

This exact implementation was imported to the new repository, as it had been concluded successful by testing, with results close to the theoretical throughput of the I<sup>2</sup>C bus. [1, p. 32].

## 3.3 Importing glue

The interface between the CSP library and the I<sup>2</sup>C drivers is not included in the open-source CSP release of 2014. Therefore, a glue is needed to pass information between the two communication layers. Giskeødegårds provided such an interface with his report, which naturally was specified to glue CSP to his implementation of I<sup>2</sup>C drivers mentioned in Section 3.2. However, the glues call to the CSP library was based on an altered version of the CSP interface, whilst only vaguely documenting these changes in his report.[4, p. 18]. The changes applied was only regarding the input parameter of the CSP method calls, in which some had been removed because they were unused. This resulted in a time consuming process of finding a bug that did not show up in any build reports or in debugging. Finally, when the source of the error was discovered, the glue was changed to function with the standard build of the CSP library interface, to prevent the same from happening in future projects.

---

<sup>1</sup>Step-by-step guide to installing CSP: <https://www.ntnu.no/wiki/display/nuts/Installing+CSP+library+for+FreeRTOS>

## 3.4 Simplified CSP API

Several fully functional and properly tested CSP source code methods for sending and receiving over CSP were presented in the EiT group report *Testing of the CSP Implementation on the NTNU Test Satellite*. [1, p. 8-18] These methods were all located in the file *CSP\_testing.c* and named with a test attribute. Some of these testing methods were rewritten to only include the code that provided its functionality, and copied to a file called *csp\_controller.c*. The idea behind the controller is to give future developers simple and abstract method calls for setting up CSP, socket listeners and transmitting data.

### 3.4.1 Create a listener

A call to the method *csp\_create\_listener()* will create a listener to the port number specified as the argument. The second argument is a pointer to the method that will become the FreeRTOS task and must be implemented by the developer. An example of a simple listener method is presented in *csp\_examples.c* and in the appendix of this report; Chapter [Source code](#), Section [CSP example code](#).

### 3.4.2 Transmit data

With the help of the controller, data can easily be transmitted to the wanted socket. By calling the method *csp\_transmit()*, a packet or several packets will be generated and sent to the socket set as the parameter. The method handles data chunks that are too big to fit into one packet, and splits it into several smaller packets which are all sent in correct order.

### 3.4.3 Packet types

The transmit method separates between three types of packets.



DATA	Used when only transmitting data
COMMAND	Used when requesting the receiver for something
DATA_AND_COMMAND	Used when requesting the receiver while also transmitting data. The command is inserted into the packet together with an ASCII enquiry character before the data, as illustrated in Figure 3.1

Listeners that needs to react to commands should be implemented to check the first byte for the ENQ character, and read the following byte as a command if the character is found.

1 byte	1 byte	0-60 bytes
ENQ	Command	Data →

Figure 3.1: Data and command sent in one packet

## 3.5 CLI control

A command line interface(CLI) simplifies software testing, as instructions can be given from a user during run-time. To control the CLI, PuTTY, a free implementation of Telnet and SSH together with a terminal, was used.[2]

### 3.5.1 CSP directory

The master branch of the software repository included a working implementation of a CLI, which had previously been added to test other sections of NUTS satellite software. To avoid spending time making a new CLI, and also, to avoid making severe changes to the already implemented CLI, CSP was implemented as a "directory" as shown in Listing 3.1.

```

-----NUTS COMMAND LINE INTERFACE-----

NUTS>  help

help    -   Displays this list
mem     -   Does a memory test
csp     -   Enter the csp "directory"

NUTS>  csp

----- Entering the CSP directory -----
CSP initialization status: SUCCESS

-----

This is module: OnBoard Computer(OBC)
i2c address: 2
-----

NUTS/CSP>  help

help    -   Displays this list
tasks   -   Displays the running FreeRTOS tasks
listen  -   Create a listener, format: listen [socketnumber]
send    -   Send a message, format: send [message]@[address]:[port]
exit    -   Exit the csp "directory"

NUTS/CSP>

```

Listing 3.1: CSP is implemented as a directory within the CLI

By accessing the CSP directory, the user will only have access to the commands within the directory, thus will not be able to run commands that are meant for testing other aspects of the satellite.

This was implemented by making minimal changes to the already existing CLI code. The changes included:

- Added a variable, called *csp\_test\_dir* that changes from *false* to *true* when the CSP directory is entered, and reset to *false* when exiting the directory.
- Added code that changes the set of CLI commands that are available when *csp\_test\_dir* is *true* to the CSP directory commands.
- Added a new shell prompt that is displayed when *csp\_test\_dir* is *true*, to let the user know which directory he/her is in.

Other directories can be added to the CLI using the same technique, which will make the CLI structural and simple even when implemented for several different purposes.

## 3.6 Example code for testing

CSP over I<sup>2</sup>C , including the glue that connects them, has already been properly tested in an EiT project for the NUTS Satellite.[1] As the new implementation is largely based off of the same code as the previous implementation, it is a reasonable assumption that the tests from the EiT project will display similar results if repeated. These tests thoroughly tested the functionality of CSP for different situations that can occur on the satellite, and presented results that were concluding CSP to be an appropriate network protocol for NUTS.

The tests created for this project are designed to give future NUTS developers a simple overview and understanding of how the CSP library can be used. If more advanced transmission options or more advanced port-listeners are to be developed, examples will be available in the source code from the EiT project tests called *CSP\_testing.c* in the *csp\_erlend* branch of the *sat-sw* repository<sup>2</sup>.

### 3.6.1 Simple transmissions

A test for verifying that the CSP implementation is able to transmit packets successfully over the I<sup>2</sup>C bus. Socket listeners and transmissions are created during run-time, making it a fast method for simple verification of functionality. For example, if a developer of the camera module has created a specific socket listener to receive commands for taking pictures/etc, he/she can transmit a specified command to the listener during run-time to verify the socket listeners functionality.

#### Socket listener

The test code uses the CLI to enable user creation of listener sockets. By running the command *listen [arg]* in the CSP directory of the CLI, a FreeRTOS task will be created as a listener to the port specified in the argument. The listener will print out the data it receives to the CLI. As this is the most simple version of a listener, it does not support rejoining of split data, which means that if a packet has been split into several packets before being

---

<sup>2</sup>Found at: [https://bitbucket.org/nuts\\_ntnu/sat-sw](https://bitbucket.org/nuts_ntnu/sat-sw) (Accessed 6.12.2014)

sent over the I<sup>2</sup>C link, then the packets will be received individually on the listener side.

## Transmitting packets

If another module wants to transmit to the socket listener created above, it could do so using the `send [arg1]@[arg2]:[arg3]` function from the CLI.

The `send` command supports packet splitting, and does so automatically if the data is of length longer than 64 bytes. To make it easier to send larger packets, without going through the hassle of typing them in to the message argument, three predefined packets have been made:

<b>ping@2:15</b>	Sends "PING" to 2:15
<b>big@2:15</b>	Sends "xxx xxxxxxxxxxxxxxxxxxxxx" to 2:15
<b>huge@2:15</b>	Sends "abcdefghijklmnopqrstuvwxyz0123456789" times 14 to 2:15

Both the *send* and the *listen* CLI commands uses the functions from the *csp\_examples.c* file. A guide for how to setup a CSP connection is included in chapter [How to use CSP](#) in the appendix in this report.

## Chapter 4

# Testing

To make sure the implementation of CSP is working correctly, it has to be tested. Although the best way to test the implementation would be on the prototype of the satellite itself, time only allowed for tests performed on a separate I<sup>2</sup>C -link setup using development boards from ATMEL to send and receive.

The satellite will most likely consist of 5 modules that will need to connect over an I<sup>2</sup>C link, the OBC, the radio, the camera, the electrical power system and the control system. Also, several of these modules will need to be able to control the flow of information over the I<sup>2</sup>C link, hence they must be able to be both a master and a slave of the network. Therefore, the tests must prove that the connection will work with more than two modules, and it must be shown that each module can control the network without manual reconfiguration of the node type.

The tests performed are simple tests that assures that basic transmissions are successfully transferred over the I<sup>2</sup>C link and received by a socket listener on the other side of the connection. More advanced and thorough tests of CSP has already been performed on the previous implementation of CSP, and are assumed to work for this implementation as well. Further testing should be performed when modules have implemented CSP to assure its correctness.

## 4.1 Test Setup

The hardware setup follows the basic setup of an I<sup>2</sup>C connection between three clients, where each of the client simulates being either the on-board computer(OBC), the radio or the camera on the satellite.

The SDA and SCL pins on the modules were connected, and each signal line were drawn to a logic high by the pullup resistors according to the I<sup>2</sup>C protocol.[13] One of the modules was used to supply voltage for the pullup resistors, and ground pins of each module were connected to assure a common ground. A full circuit scheme is illustrated in Figure 4.1

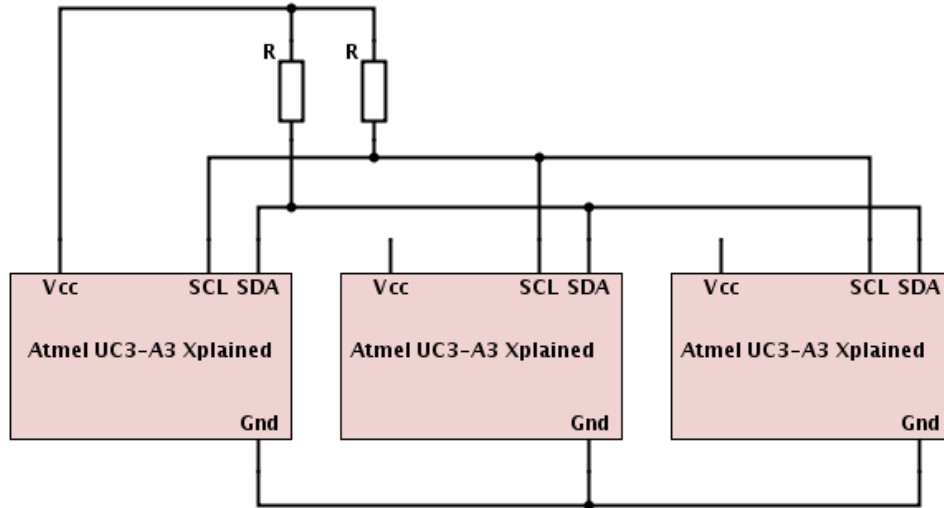


Figure 4.1: Schematics of the test setup

### 4.1.1 Hardware

The hardware components below were used to perform the tests

3 x	<b>Atmel UC3-A3 Xplained</b>	Development boards with I <sup>2</sup> C support. Uses the same microprocessor as on the satellite.
2 x	<b>2.7 k<math>\Omega</math> resistors</b>	Pullup resistors for the I <sup>2</sup> C link.
1 x	<b>Project board</b>	For easy connection.
10 x	<b>Cables</b>	Jumper cables used to connect Xplained boards to the project board
1 x	<b>Atmel AVR Dragon</b>	Debugger.
1 x	<b>RS232 to USB cable</b>	To communicate with the CLI
1 x	<b>Serial to RS232 converter</b>	Circuit to connect the RS232 to the XPlained
1 x	<b>Saleae Logic 16</b>	Software for analysing transmission links.

## 4.2 Tests

The tests consisted of verifying that several CSP modules could be initiated and able to communicate over the same I<sup>2</sup>C link. It was decided that  $N \geq 3$  would be sufficient to conclude for the success of plural modules, and their functionality was tested by transmitting from each module to the others, and verify that the packets were received correctly.

### 4.2.1 Simple data transfers

This test verifies that socket listeners are successfully receiving CSP packets over the I<sup>2</sup>C link. Transfers are initiated by calling the *send* function from

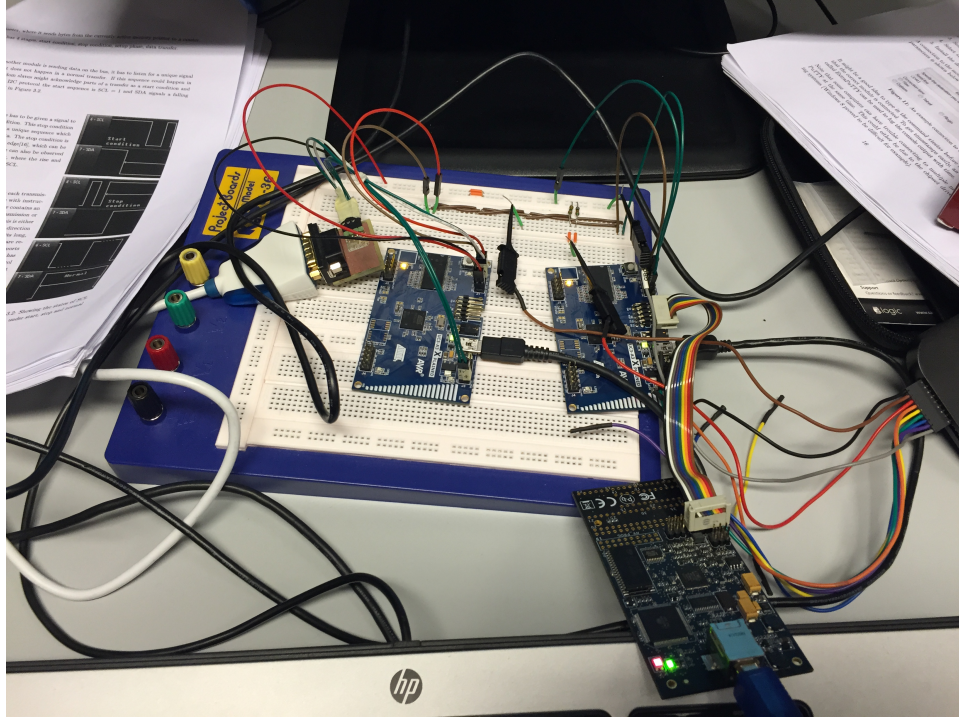


Figure 4.2: Setup of the testing environment, with two client modules

the CLI, and the content of the data packet is printed to the CLI on the receiving end.

#### 4.2.2 Splitting of data

The test displays the functionality of packet splitting for larger data chunks. Calling the `csp_transmit()` method from the `csp_controller` with a pointer to data larger than 64 bytes, will automatically split the data into 64 byte packets before sending them off individually. This is verified by the listener printing the individual packets on the receiving end. This test should also be performed with receiving listeners capable of recombining the split data, but this is not implemented for the current test.



### 4.2.3 Loop test

The test verifies that modules are able to rapidly switch between being a master or a slave of the I<sup>2</sup>C connection. A packet containing data is looped between the three modules(OBC, radio and camera), and the number of successful transfers is printed to the CLI terminal after each received packet. The test is initiated by the OBC transmitting to the radio module, which forwards the data to the camera. The camera passes the packet to the OBC, and a loop has been created. The test runs until it is powered off, or until it stops due to an error.

The test was ran for 2 hours with 200 ms delay between receiving and transmitting a packet for each module, yielding somewhat lower than 1.67 loops per second due to additional processing time.

# Chapter 5

## Results

### 5.1 Using CLI to test

The CLI's CSP directory implementation provided an easy way of running tests, without making the already implemented CLI overpopulated with commands only used for CSP testing. Run-time configurations of CSP addresses and initializations facilitated simple and quick setups of the different modules, as opposed to hard coding the addresses and socket listeners in the program code.

### 5.2 Test results

#### 5.2.1 Simple transmissions

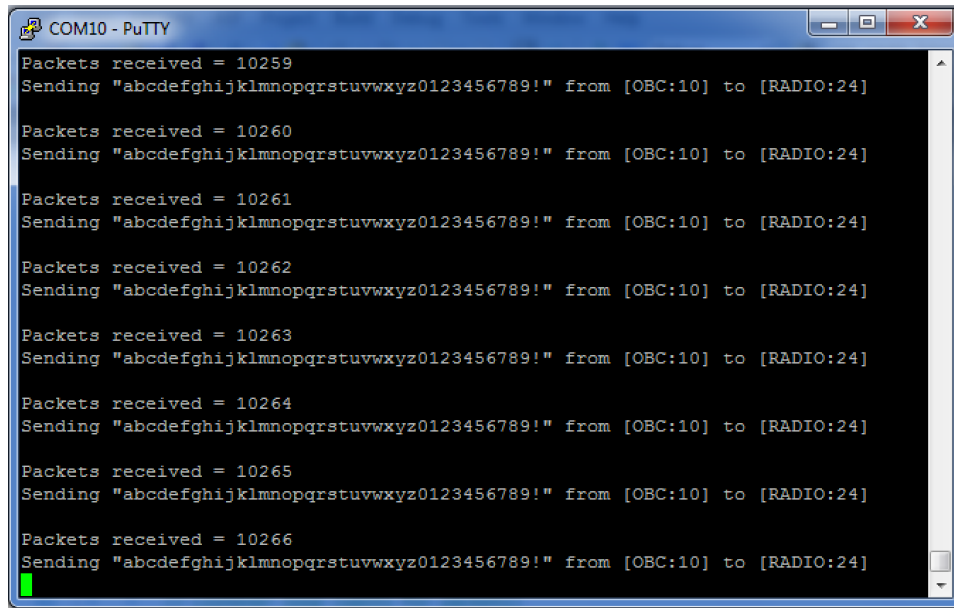
The tests showed that packets of data was successfully transmitted to the specified sockets, and the received data was printed to the CLI terminal on the receiving module. When trying to transmit to addresses that were not reachable on the bus, or when the bus was busy due to other transmissions, the glue layer provided an error message, but the CSP library remained silent about the error, as reported by the EiT project.[1, p. 29]

### 5.2.2 Splitting of data chunks

A large chunk of data was sent from one module to another. On the receiving module, the data was proved to be split into several smaller packets and was received sequentially in correct order. Although this is the desired functionality, it would have been preferable to include some checks for incorrect order of the received data.

### 5.2.3 Loop test results

After running for 2 hours, the packet had been looped successfully between the three modules 10266 times without a single failure, as shown in Figure 5.1.



```
COM10 - PuTTY
Packets received = 10259
Sending "abcdefghijklmnopqrstuvwxyz0123456789!" from [OBC:10] to [RADIO:24]

Packets received = 10260
Sending "abcdefghijklmnopqrstuvwxyz0123456789!" from [OBC:10] to [RADIO:24]

Packets received = 10261
Sending "abcdefghijklmnopqrstuvwxyz0123456789!" from [OBC:10] to [RADIO:24]

Packets received = 10262
Sending "abcdefghijklmnopqrstuvwxyz0123456789!" from [OBC:10] to [RADIO:24]

Packets received = 10263
Sending "abcdefghijklmnopqrstuvwxyz0123456789!" from [OBC:10] to [RADIO:24]

Packets received = 10264
Sending "abcdefghijklmnopqrstuvwxyz0123456789!" from [OBC:10] to [RADIO:24]

Packets received = 10265
Sending "abcdefghijklmnopqrstuvwxyz0123456789!" from [OBC:10] to [RADIO:24]

Packets received = 10266
Sending "abcdefghijklmnopqrstuvwxyz0123456789!" from [OBC:10] to [RADIO:24]
```

Figure 5.1: A screen capture of the CLI after running the loop test for 2 hours

## 5.3 Issues

Although the CSP library has been tested before, some new issues were discovered during the testing of the implementation. In addition, some of the issues that were mentioned in the report by the EiT group were revisited.

### 5.3.1 Initializing CSP

A possible bug in the compiled CSP library was detected in the early stage of testing. One of the initialization methods, *csp\_i2c\_init()*, from the *csp-if-i2c.c* source file in the library, is shown in listing 5.1

---

```
int csp_i2c_init(uint8_t addr, int handle, int speed) {
    /* Create i2c_handle */
    csp_i2c_handle = handle;
    if (i2c_init(csp_i2c_handle, I2C_MASTER, addr, speed, 10
                ,10, csp_i2c_rx) != E_NO_ERR)
        return CSP_ERR_DRIVER;

    /* Register interface */
    csp_route_add_if(&csp_if_i2c);

    return CSP_ERR_NONE;
}
```

---

Listing 5.1: I<sup>2</sup>C initialization method in *csp-if-i2c.c*

The method returns '1', which is not the definition of either `CSP_ERR_DRIVER` or `CSP_ERR_NONE`. As the library is compiled and referenced to as a static library file, debugging in run-time was somewhat troublesome, and did not conclude in any solutions.

The problem is believed to be the `E_NO_ERR` value, which is possibly incorrectly defined. CSP operates with 0's for successes, and negative numbers for errors. Listing 5.2 is a direct copy of the definition of `E_NO_ERR` from the CSP library, and shows that even the developers have recognized this as a possible bug, but have not yet corrected it.

---

```

/**
 * The return value of the driver is a bit strange,
 * It should return E_NO_ERR if successfull and the value is -1
 */
#define E_NO_ERR    -1

```

---

Listing 5.2: Possible bug found in *csp-if-i2c.h*

To solve this issue, the return value of the *csp\_i2c\_init()* was tested to be greater or equal to '0', instead of equal to '0'. There has been no indications of the I<sup>2</sup>C not initializing properly when returning '1', but in fact, '-11', which is the defined CSP\_ERR\_DRIVER value, was received when the I<sup>2</sup>C was not properly implemented.

### 5.3.2 Return values not propagating through the CSP library

The report *Testing of the CSP Implementation on the NTNU Test Satellite* presented an issue regarding the return values of the transmission calls.[1, p. 29] If a transmission is unsuccessful, no error value is returned from the function calling the CSP transmission, hence leaving the caller of the function in belief that the transmission was successful. This is of course a major problem, as error handling is an important part of developing software for space environments.

The issue was investigated, but no immediate solution was found. It is assured that the glue layer returns negative values to the CSP library when transmissions are unsuccessful, but the error does not propagate through the library and back to the caller function properly.

### 5.3.3 Missing input parameters to CSP functions

During testing it was discovered that Andreas Giskegårds implementation of the I<sup>2</sup>C to CSP glue did not function properly with a new build of the CSP library. While the new implementation could receive information from the old implementation successfully, it was not able to transmit properly. The sending function returned a NACK error, indicating that no slave was receiving the message it was trying to send.[13, p. 10] No errors showed up during debugging that would explain why it did not function, which made

the process of finding the error time consuming and difficult.

As the new implementation was able to receive messages, and because the old implementation functioned perfectly over the I<sup>2</sup>C test hardware, the problem was assumed to be software related. When debugging, it was found that the correct addresses and port number was passed into the CSP send function, but as the CSP library is implemented as a static library, debugging into this source code was not feasible.

A logic analyser from Saleae was used to be able to see what was, if anything, actually being sent over the I<sup>2</sup>C link during an unsuccessful transmission with a NACK error. The following illustrations show analysis of a successful transmission from old to new implementation, and an unsuccessful transmission from new to old implementation.

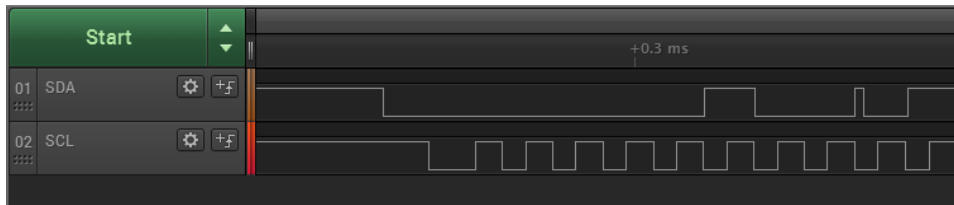


Figure 5.2: A successful transmission of the first 8 bits of a packet

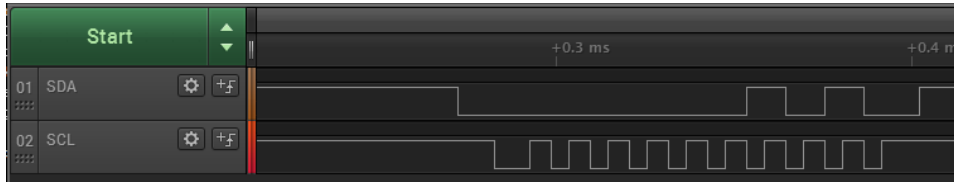


Figure 5.3: An unsuccessful transmission of the first 8 bits of a packet

As seen in Figures 5.2 and 5.3, the working and not working transmissions differ in two ways.

Firstly, the SDA signal is pulled low by the slave directly after the master has set it high after the eighth bit. This signalizes that the slave has received the bits, and that the master can continue transmitting data. In the unsuccessful transfer on the other hand, the ACK bit is never pulled low by any slave. This forces the master to stop the transmission, and explains why the NACK error was returned in the software.

Secondly, at first glance, the addresses in the transmissions seems to be '4' in Figure 5.2 and '2' in 5.3 which was the addresses used in the tests. But with further research, it was discovered that the first byte of a I<sup>2</sup>C transfer consists of the receiving address(7 bits) followed by a bit stating the direction of the transmission.[9]. This results in addresses '2' in Figure 5.2 and '1' in 5.3, which gave a natural explanation to the problem.

As correct addresses were passed into the CSP library, the bug either had to be from the library itself, or in the interface between the glue and the library. Here it was found that the old implementation had removed some parameters from some of the functions in the CSP library, resulting in wrongly passing information to the functions when applying the old glue code to a new build of the library and its interface methods. After this was handled, the CSP worked as it was supposed to.

## Chapter 6

# Discussion

### 6.1 Application layer protocol options

One of the main benefits of using CSP is the broad possibility of payload formats it can carry. As data is transmitted to specified ports on each module, application layer protocols can be designed individually for each application. This would not have been the case if a network layer had been removed, as there would be only one socket listener for each module, that would have to handle all incoming packets. Instead, there would have to be implemented another way of separating data based on its source and content. As implementations of many sections of the satellite has already been started, by many different students, they now have the possibility to implement communication handlers for their specific purpose, without deciding on a common protocol for all data transfers. Still, it might be considered best practice to use similar structures, or at least keep proper documentation of each application protocol.

The implemented CSP controller supports splitting of packets when larger data chunks is transmitted, and performs the splitting automatically, further simplifying development. Rejoining the packets on the receiving end is left to each individual developer, but examples of this are available in the source code.<sup>1</sup>

---

<sup>1</sup>Examples can be found in the *CSP\_testing.c* source file in the *csp\_erlend* branch of the *sat\_sw* repository



## 6.2 I<sup>2</sup>C without CSP

Some of the sensors on the satellite are designed to communicate with modules over the same I<sup>2</sup>C bus as the one being used for CSP transmissions. This means that some of the communication on the link will be with the CSP protocol, while calls to sensors without CSP implemented will happen without the network protocol. While certainly possible to implement, adjustment to the I<sup>2</sup>C code and possibly structural changes will be needed, as all incoming packets are regarded as CSP packets with the current I<sup>2</sup>C implementation. How this can be implemented should be looked further into, but is not regarded as within scope of this project.

## Chapter 7

# Conclusion

The project is concluded to be partly successful. CSP was implemented and tested for its basic functionality, while the I<sup>2</sup>C driver and CSP-to-I<sup>2</sup>C glue imported from earlier NUTS projects was edited to function successfully within the new software repository. The tests showed successful results when testing for expected behavior of the system. However, there are few precautions taken to handle unexpected errors in a safe manner, which is necessary in the harsh and uncontrollable environment the satellite will operate in.

A lot of time was spent debugging issues that could have been detected at an earlier stage with a more thorough reading of the report *Implementing CSP over I2C on the NTNU Test Satellite*. Hence, more time would have been available to investigate the already discovered issues of CSP, as these issues are still not solved. Also, a better CSP controller with more advanced functionality could have been designed and implemented, providing simpler setup and use of CSP for future NUTS software developers.

Regardless, it is now possible for NUTS students to start integrating CSP in their implementations. A framework for testing has been added with the simplicity of a CLI, and examples showing basic setups and functionality has been presented. The communication structure is not yet entirely completed, but is sufficient for the purpose of facilitating further development.

# References

- [1] Blakkisrud J. Delabahan C. Munthe-Kaas N. Bjørnevik, A. and Ø. Sture. Testing of the CSP implementation on the NTNU Test Satellite. Experts in Teams, Project report, NTNU, Trondheim, 2014.
- [2] Chiark. PuTTY, A Free Telnet/SSH Client. [Online]. <http://www.chiark.greenend.org.uk/~sgtatham/putty/> (accessed 15.12.2014).
- [3] FreeRTOS. Official website. [Online]. <http://www.freertos.org> (accessed 15.10.2014).
- [4] Andreas Giskeødegård. Implementing CSP over I2C on NTNU Test Satellite. Project report, NTNU, Trondheim, 2012.
- [5] GNU. GNU LESSER GENERAL PUBLIC LICENSE. [Online]. <https://www.gnu.org/licenses/lgpl.html> (accessed 30.10.2014).
- [6] GomSpace. CubeSat Space Protocol. [Online]. <http://gomspace.com> (accessed 22.09.2014).
- [7] GomSpace. CubeSat Space Protocol (CSP). [Online]. <http://www.gomspace.com/documents/GS-CSP-1.1.pdf> (accessed 25.11.2014).
- [8] GomSpace. libcsp on Github. [Online]. <https://github.com/GomSpace/libcsp> (accessed 23.09.2014).
- [9] I2C-org. I2C-Bus, What's that? [Online]. <http://www.i2c-bus.org/i2c-bus/> (accessed 17.11.2014).
- [10] i2c.info. I2C Info – I2C Bus, Interface and Protocol. [Online]. <http://i2c.info> (accessed 17.11.2014).

- [11] Nasa. CubeSat Launch Initiative. [Online]. [http://www.nasa.gov/directorates/heo/home/CubeSats\\_initiative.html#.VD0zdkuSMrs](http://www.nasa.gov/directorates/heo/home/CubeSats_initiative.html#.VD0zdkuSMrs) (accessed 28.09.2014).
- [12] NXP. NXP Introduces Industry's First I2C-Bus Controllers Supporting New Ultra Fast-Mode Specification. [Online]. <http://www.nxp.com/news/press-releases/2012/04/nxp-introduces-industrys-first-i2c-bus-controllers-supporting-new-ultra-fast-mode-specification.html> (accessed 17.11.2014).
- [13] NXP. UM10204 - I2C-bus specification and user manual. [Online]. [http://www.nxp.com/documents/user\\_manual/UM10204.pdf](http://www.nxp.com/documents/user_manual/UM10204.pdf) (accessed 17.11.2014).
- [14] Philip P. Shirvani and Edward J. McCluskey. Fault-Tolerant Systems in A Space Environment. The CRC ARGOS Project, Stanford University, San Fransisco, USA, 1998.
- [15] Waf. Waf - Project Summary. [Online]. <https://www.openhub.net/p/waf> (accessed 20.11.2014).
- [16] Wikipedia. NUTS 1 (satellite). [Online]. [http://en.wikipedia.org/wiki/NUTS\\_1\\_\(satellite\)](http://en.wikipedia.org/wiki/NUTS_1_(satellite)) (accessed 28.09.2014).

# List of Figures

1.1	Implementing CSP over I <sup>2</sup> C for the NUTS satellite . . . . .	2
2.1	CSP header structure . . . . .	4
2.2	The illustration shows how all modules on the satellite is connected through a backplane. . . . .	5
2.3	The physical structure of an I <sup>2</sup> C link. . . . .	6
2.4	Initiation of a transmission . . . . .	7
3.1	Data and command sent in one packet . . . . .	12
4.1	Schematics of the test setup . . . . .	17
4.2	Setup of the testing environment, with two client modules . .	19
5.1	A screen capture of the CLI after running the loop test for 2 hours . . . . .	22
5.2	A successful transmission of the first 8 bits of a packet . . . .	25
5.3	An unsuccessful transmission of the first 8 bits of a packet . .	25

## Appendix A

# Step-by-step guide to installing CSP

GomSpace, the provider of CSP libraries, uses the **waf** build system to configure and compile CSP for the different systems. The **waf** system is not very intuitive, and there is not much useful information about how to use it for the CSP system online either. To save time for future NUTS-students using CSP, a step-by-step guide to download, configure and install CSP with I<sup>2</sup>C support for FreeRTOS is presented below. The guide is based upon GomSpace documentation and the work of former NTNU- and NUTS-student Andreas Giskeødegård presented in the thesis *Implementing CSP over I<sup>2</sup>C on NTNU Test Satellite*[4].

**NOTE:** This guide shows how to install CSP for FreeRTOS running on an ATMEL UC3A3256 microcontroller. Other configurations will require a different installation.

### Step 1 - Download libcsp-master.zip

The files needed to install CSP is located at GomSpace's Github<sup>1</sup>. Download the .zip and extract it to a project appropriate location.

### Step 2 - Specify the part, architecture and board to compile for

In the extracted folder, open the PYTHON script **wscript**. Add these three

---

<sup>1</sup><https://github.com/GomSpace/libcsp/tree/master>

variables to the CFLAGS:

- '-mpart=uc3a3256'
- '-DBOARD=USER\_BOARD'
- '-ARCH=ucr2'

It should look something like this:

```
# Setup CFLAGS
if (len(ctx.env.CFLAGS) == 0):
    ctx.env.prepend_value('CFLAGS', ['-DBOARD=USER_BOARD', '-mpart=uc3a3256', '-ARCH=ucr2', '-Os', '-Wall', '-g', '-std=gnu99'])
```

### Step 3 - Setting up Python

First, make sure PYTHON is installed on your computer. It will be necessary to run the **waf** script in the cmd-console, so include PYTHON as an environment variable<sup>2</sup> to make it accessible from all locations. Open the cmd-console and navigate to the extracted libesp-master directory.

### Step 4 - Add the AVR32-gcc compiler as environment variable

Add the path to the AVR32 compiler to PATH in the environment variables the same way as PYTHON was added. If Atmel Studio 6 was installed to the default path, then the path to copy into the PATH environment variable should be: C:\Program Files\Atmel\Atmel Toolchain\AVR32 GCC\Native\3.4.1057\avr32-gnu-toolchain\bin

### Step 5 - Configuring the build

Before we can build the library, we need to configure how we want it to be built. This is done using the **python waf configure** command.

There are several configurations we need to set in order to correctly build the esp library:

- Toolchain** Set which toolchain to use for compilation
- Prefix** Location to place the built libraries
- Bus-system** Enabling of bus-system drivers. The satellite uses I<sup>2</sup>C , therefore, we use --enable-if-i2c
- Queue length** Set the length of the queue

---

<sup>2</sup><https://docs.python.org/2/using/windows.html> - Section 3.3.1

**-OS** Set the OS you are compiling CSP for with `--with-os=OS`

- Use `--with-os=freertos` to compile for FreeRTOS
- Use `--with-freertos=PATH` to set the path to the FreeRTOS.h file in your project

**-Includes** Set all necessary includes with `--includes=PATHS`

- Insert path to all project folders containing header files used in the project. Separate the paths with commas.

**NOTE:** To avoid character encoding errors, type the text directly into the cmd-console, and do not copy and paste it from somewhere else (especially not the double dashes "--").

A complete configure command may look something like this:

```
Microsoft Windows [Version 6.1.7601]
Copyright (c) 2009 Microsoft Corporation. All rights reserved.

C:/Users/Public/Documents/sat-sw/libcsp-master>python waf configure
--toolchain=avr32 --prefix=../install --jobs=1 --enable-if-i2c --w
ith-router-queue-length=30 --with-os=freertos --with-freertos=../sa
t-sw/src/freertos/include --includes=../sat-sw,../sat-sw/src,../sat
-sw/src/freertos/include,../sat-sw/src/freertos/portable/GCC/AVR32_
UC3,../sat-sw/src/freertos,../sat-sw/src/FreeRTOS-Plus-Trace/Includ
e,../sat-sw/src/FreeRTOS-Plus-Trace/Configuration,../sat-sw/src/ASF
/common/utils,../sat-sw/src/ASF/common/boards,../sat-sw/src/ASF/com
mon/boards/user_board,../sat-sw/src/ASF/common/services/clock,../sa
t-sw/src/ASF/common/services/clock/uc3a3_a4,../sat-sw/src/ASF/avr32
/utils/startup,../sat-sw/src/ASF/avr32/utils/header_files,../sat-sw
/src/config,../sat-sw/src/ASF/avr32/utils,../sat-sw/src/ASF/avr32/u
tils/preprocessor,../sat-sw/src/ASF/avr32/drivers/,../sat-sw/src/AS
F/avr32/drivers/flashc,../sat-sw/src/ASF/avr32/drivers/gpio,../sat-
sw/src/ASF/avr32/drivers/tc,../sat-sw/src/ASF/avr32/drivers/pm,../s
at-sw/src/ASF/avr32/drivers/intc,../sat-sw/src/ASF/thirdparty/newli
b_addons/libs/include,"C:/Program Files/Atmel/Atmel Toolchain/AVR32
GCC/Native/3.4.1057/avr32-gnu-toolchain/avr32/include/avr32","C:/Pr
ogram Files/Atmel/Atmel Toolchain/AVR32 GCC/Native/3.4.1057/avr32-g
nu-toolchain/avr32/include"
```

When ran, it should result in the following:

```
Setting top to : C:\Users\Public\Documents\sat-sw\libcsp-master
Setting out to : C:\Users\Public\Documents\sat-sw\libcsp-master\build
Checking for program gcc,cc : avr32-gcc
Checking for program ar : avr32-ar
Checking for program size : avr32-size
Checking for endianness : big
Checking for header stdbool.h : yes
'configure' finished successfully (0.211s)
```



## Step 6 - Building the library

When CSP is configured successfully, it is time to build and install it. To do this, use the command **python waf build install**. If everything works, it should look something like this:

```
C:\Users\Public\Documents\saat-sw\libcsp-master>python waf build install
[ 1/191] c: src\arch\freertos\csp_malloc.c -> build\src\arch\freertos\csp_malloc.c.1.o
[ 2/191] c: src\arch\freertos\csp_queue.c -> build\src\arch\freertos\csp_queue.c.1.o
[ 3/191] c: src\arch\freertos\csp_semaphore.c -> build\src\arch\freertos\csp_semaphore.c.1.o
[ 4/191] c: src\arch\freertos\csp_system.c -> build\src\arch\freertos\csp_system.c.1.o
..src\arch\freertos\csp_system.c: In function 'csp_sys_tasklist':
..src\arch\freertos\csp_system.c:430: warning: pointer targets in passing argument 1 of 'vTaskList' differ in signedness
C:\Users\Public\Documents\saat-sw\saat-sw\src\freertos\include\task.h:1305: note: expected 'char *' but argument is of type 'signed
char *'
[ 5/191] c: src\arch\freertos\csp_thread.c -> build\src\arch\freertos\csp_thread.c.1.o
..src\arch\freertos\csp_thread.c: In function 'csp_thread_create':
..src\arch\freertos\csp_thread.c:33: warning: pointer targets in passing argument 2 of 'xTaskGenericCreate' differ in signedness
C:\Users\Public\Documents\saat-sw\saat-sw\src\freertos\include\task.h:1516: note: expected 'const char * const' but argument is of t
ype 'const signed char * const'
[ 6/191] c: src\arch\freertos\csp_time.c -> build\src\arch\freertos\csp_time.c.1.o
[ 7/191] c: src\csp_buffer.c -> build\src\csp_buffer.c.1.o
[ 8/191] c: src\csp_conn.c -> build\src\csp_conn.c.1.o
[ 9/191] c: src\csp_endian.c -> build\src\csp_endian.c.1.o
[10/191] c: src\csp_io.c -> build\src\csp_io.c.1.o
[11/191] c: src\csp_port.c -> build\src\csp_port.c.1.o
[12/191] c: src\csp_route.c -> build\src\csp_route.c.1.o
[13/191] c: src\csp_service_handler.c -> build\src\csp_service_handler.c.1.o
[14/191] c: src\csp_services.c -> build\src\csp_services.c.1.o
[15/191] c: src\csp_sfp.c -> build\src\csp_sfp.c.1.o
[16/191] c: src\interfaces\csp_if_i2c.c -> build\src\interfaces\csp_if_i2c.c.1.o
[17/191] c: src\interfaces\csp_if_lo.c -> build\src\interfaces\csp_if_lo.c.1.o
[18/191] c: src\transport\csp_udp.c -> build\src\transport\csp_udp.c.1.o
[19/191] c: src\lib: build\src\arch\freertos\csp_malloc.c.1.o build\src\arch\freertos\csp_queue.c.1.o build\src\arch\freertos\csp_semaphore.c.1.o
build\src\csp_buffer.c.1.o build\src\csp_conn.c.1.o build\src\csp_endian.c.1.o build\src\csp_io.c.1.o build\src\csp_port.c.1.o b
uild\src\csp_route.c.1.o build\src\csp_service_handler.c.1.o build\src\csp_services.c.1.o build\src\csp_sfp.c.1.o build\src\interf
aces\csp_if_i2c.c.1.o build\src\interfaces\csp_if_lo.c.1.o build\src\transport\csp_udp.c.1.o -> build\libcsp.a
[ 20/191] Leaving directory 'C:\Users\Public\Documents\saat-sw\libcsp-master\build'
waf: finished successfully (4.024s)
waf: Entering directory 'C:\Users\Public\Documents\saat-sw\libcsp-master\build'
* install C:\Users\Public\Documents\saat-sw\csp-install\include\csp\csp.h (from include\csp\csp.h)
* install C:\Users\Public\Documents\saat-sw\csp-install\include\csp\csp_buffer.h (from include\csp\csp_buffer.h)
* install C:\Users\Public\Documents\saat-sw\csp-install\include\csp\csp_conn.h (from include\csp\csp_conn.h)
* install C:\Users\Public\Documents\saat-sw\csp-install\include\csp\csp_endian.h (from include\csp\csp_endian.h)
* install C:\Users\Public\Documents\saat-sw\csp-install\include\csp\csp_debug.h (from include\csp\csp_debug.h)
* install C:\Users\Public\Documents\saat-sw\csp-install\include\csp\csp_endian.h (from include\csp\csp_endian.h)
* install C:\Users\Public\Documents\saat-sw\csp-install\include\csp\csp_interface.h (from include\csp\csp_interface.h)
* install C:\Users\Public\Documents\saat-sw\csp-install\include\csp\csp_platform.h (from include\csp\csp_platform.h)
* install C:\Users\Public\Documents\saat-sw\csp-install\include\csp\interfaces\csp_if_lo.h (from include\csp\interfaces\csp_if_lo.h)
* install C:\Users\Public\Documents\saat-sw\csp-install\include\csp\interfaces\csp_if_i2c.h (from include\csp\interfaces\csp_if_i2c
.h)
* install C:\Users\Public\Documents\saat-sw\csp-install\include\csp\csp_autocfg.h (from build\include\csp\csp_autocfg.h)
* install C:\Users\Public\Documents\saat-sw\csp-install\lib\libcsp.a (from build\libcsp.a)
waf: Leaving directory 'C:\Users\Public\Documents\saat-sw\libcsp-master\build'
waf: finished successfully (0.061s)
```

## Troubleshoot

As the **waf** building system contains some bugs, and in general does not always behave the way it should, I've listed some of the workarounds.

1. **The configure script cannot determine the version of gcc/avr32-gcc on your computer**
  - (a) Are you sure you have typed the **--toolchain=avr32-** directly into the cmd-console during configuration?
  - (b) Is the path to the avr32-gcc compiler an environment variable?
2. **When running build install, the compiled libraries are never saved to the location specified by the --prefix=PATH**
  - (a) Try configuring, building and installing all in one call, like this:

```
Microsoft Windows [Version 6.1.7601]
Copyright (c) 2009 Microsoft Corporation. All rights reserved.

C:/libcsp-master>python waf configure (+all configurations
) build install
```

### 3. The build crashes unexpectedly

- (a) Issues like this has been solved earlier by adding `--disable-verbose` `--disable-output` to the configuration. These options mainly disables some debugging options and are not directly relevant to the compiled libraries.
- (b) Adding `--jobs=1` to the configuration may also help.

## Appendix B

# How to use CSP

A simple example of how CSP can be used is shown in the *csp\_examples.c* file from the source code in the *csp\_erlend* branch of the *sat-sw* repository on Bitbucket.<sup>1</sup> It should be noticed that this might be located differently when you read this, as branches are merged with the master branch upon completion. Anyways, this "how to" guide will show how to set up a simple CSP socket listener, and how to transmit messages.

In simple terms, a CSP socket listener will be implemented as a FreeRTOS task listening to a specific port on a specific CSP/I<sup>2</sup>C address. A blocking CSP function named *csp\_read()* will return a **struct** containing the packet information if a packet has been received over the I<sup>2</sup>C link with the address and port number of the listener socket. The **struct** contains all necessary variables such as:

<b>id</b>	A struct containing information such as the source and destination ports and addresses
<b>length</b>	The number of data bytes in the packet
<b>data</b>	A pointer to the data

### Create a new module

Add a module and its info to the *csp\_controller.h* file from the *csp\_erlend* branch. The process contains four steps. Use the examples already added to the file for guidance.

---

<sup>1</sup>Available at: [https://bitbucket.org/nuts\\_ntnu/sat-sw](https://bitbucket.org/nuts_ntnu/sat-sw) (Accessed 5.12.2014)

1. Add your new module with an appropriate name to the module definitions at the top under *module definitions*
2. Add a unused CSP and I<sup>2</sup>C addresses to your module. Addresses range from 0 to 31
3. Add your module to the list of **struct** *nuts\_modules* and fill in the required info.
4. The *this\_variable* must somehow be set to the module you want to use. The *init [arg]* CLI-command can be used to do this during runtime, which is useful when testing, but eventually, you will want to make this a constant variable.

The *csp\_controller* .h and .c files should be exactly the same for all implementations of CSP for NUTS. This makes it possible for all CSP modules to reference each other abstractly and easily.

### Set up a listener

Now that your module is created, lets create a listener that will receive incoming packets. This process consists of two steps.

First, create a function that will work as the listener. The code below describes the simplest form of a socket listener, and can be used as base for further development.

---

```
void example_listen_task(void *ptr) {
    csp_conn_t *conn;
    uint8_t listen_socket = *(uint8_t*)ptr;

    csp_socket_t *sock = create_listen_socket(listen_socket);
    if (sock == NULL) {
        printf("\rError creating/binding port/socket.\r\n");
        while(true); // Stay in task in order to not crash FreeRTOS
    }

    while(true) {
        conn = csp_accept(sock, CSP_ACCEPT_TIMEOUT);

        // Listener loop (waits for packets)
        while(true){
            csp_packet_t * packet = csp_read(conn, CSP_READ_TIMEOUT);
            if (packet == NULL){
                csp_close(conn);
                break;
            }
        }
    }
}
```

```
        printf("Packet received!\\r\\n");
    }
    if (conn != NULL)
        csp_close(conn);
    }
}
```

---

Second, start the listener by calling the *csp\_create\_listener* function from *csp\_controller.c*. The function takes two arguments, the port number and a pointer to the function you created in step one.

---

```
#define MY_PORT_NUMBER        0x10

csp_create_listener(MY_PORT_NUMBER, &example_listen_task);
```

---

That's it, easy peasy.

**NOTE:** Review the source file *csp\_testing.c* located in the *CSP\_testing* directory in the *csp\_erlend* branch for a lot of examples of more advanced listeners, included packet rejoining, command handling and more!

## Transmitting data

Now to the fun part, sending (and hopefully receiving!) data.

The *csp\_controller.c* contains a function called *csp\_transmit* that takes seven arguments:

---

```
int csp_transmit(packet_type_t type,
                 module_socket_t *dest,
                 module_socket_t *source,
                 char *data,
                 int size,
                 command_t command,
                 csp_prio_t priority)
```

---

Some of the parameters are self explanatory, such as dest and source (csp addresses), the data pointer, and the data size. The other ones might be beneficial to explain in further detail:

<b>type</b>	Can be set to DATA, COMMAND or COMMAND_PLUS_DATA depending on what you are transmitting.
<b>command</b>	Set the command associated with the data. Some commands are already defined in <i>csp_controller.h</i> , but add more if needed.
<b>priority</b>	Set the priority of the data from 0 to 3, 0 being the highest priority. Transmissions with higher(lower number) priority will be sent before lower(higher number)priority packets

Calling this function is all you need to transmit successfully. The function will split packets automatically if the data exceeds the 60 byte limit, but you will need to include some extra processing at the listener side to rejoin it. A split packet is recognized by receiving a SPLIT\_START command before receiving the split packets, and a SPLIT\_STOP command when all the split data has been transmitted.

**NOTE:** For a simple way to see if your socket listener works, use the CLI command *send [message@address:port]* which will transmit the message to the specified port at the address.

Example:

```
-----NUTS COMMAND LINE INTERFACE-----
-----
This is module: RADIO
i2c and CSP address: 4
-----

NUTS/CSP> send Hello world!@2:15

Message: Hello world! was sent to [OBC:15]

NUTS/CSP>
```

# Appendix C

## Source code

### C.1 CSP CLI interface

#### C.1.1 csptests.h

---

```
/*
 * csptests.h
 *
 * Created: 07-Nov-14 8:17:06 PM
 * Author: Erlend
 */

#ifndef CSPTESTS_H_
#define CSPTESTS_H_

typedef void(*csp_shell_function_t)(const char* args);

typedef struct xCLL_COMMAND_DEFINITION {
    /* Command input string, the command the user enters to start a program, e.g 'help' */
    const char * const pcCommand;
    /* A string that describes the command and it's usage */
    const char * const pcHelpString;
    /* A pointer to the function that implements the command */
    const csp_shell_function_t pxCommandInterpreter;
} CSP_cli_command_definition_t;

void taskCmd(const char* pcCommandString);

int csp_run_command(char* command, char* args);
void printModuleInfo(void);
//void cspPrintCommands(const char* args);
void cspEnter(const char* args);

#endif /* CSPTESTS_H_ */
```

---

#### C.1.2 csptests.c

---

```

/*
 * csptests.c
 *
 * Created: 10-Nov-14 3:08:03 PM
 * Author: Erlend
 */
#include "csptests.h"
#include "csplib/CSP_testing/csp_testing.h"
#include "cli/cli.h"
#include "FreeRTOS.h"
#include "task.h"
#include "csplib/csp_controller.h"
#include "csplib/csp_example.h"

void cspExit(const char* args);
void cspPrintCommands(const char* args);
void setupCSP(portCHAR * str);
//static void cspPrintCommands(const char* args);

static xTaskHandle xTaskListenTest = NULL;

static CSP_cli_command_definition_t prvCmdTable[] = {
    {"help", "Print a list of all CSP commands", &cspPrintCommands}
    , {"init", "CSP setup: init [OBC/RADIO/CAMERA/etc]", &setupCSP}
    , {"tests", "Lists all available tests", &test_print_available_tests}
    , {"run", "Runs test [argument]", &test_start_test_task}
    , {"verbose", "Toggle debug text output on/off.", &test_toggle_verbosity}
    , {"memory", "Print CSP memory info.", &test_print_csp_memory}
    , {"tasks", "Lists current running tasks in a tabular format", &taskCmd}
    , {"prec", "Print how many packets this module has received", &print_packets_received}
    , {"loopt", "Loop a packet between OBC, RADIO and CAMERA for testing", &example_start_loop_test}
    , {"listen", "Start new listener to port [argument]", &example_setup_of_listener}
    , {"send", "Send messages, format: message@address:port", &example_send_simple_message}
    , {"exit", "Leave the CSP \"directory\"", &cspExit}
};

void cspEnter(const char* args){
    printf("\r\n-----\r\n");
    csp_test_dir = true;
}

void printModuleInfo(void){
    printf("\r\n-----\r\n");
    printf("This is module: %s\r\n", nuts_modules[this_module].name);
    printf("CSP-addr: %i I2C-addr: %i\r\n", nuts_modules[this_module].csp_address,
        nuts_modules[this_module].i2c_address);
    printf("Default listen port: %i\r\n", nuts_modules[this_module].default_listen_port);
    printf("-----\r\n");
}

void setupCSP(portCHAR * str){
    int i = 0;
    Bool found = false;
    int number_of_modules = sizeof(nuts_modules)/sizeof(nuts_module_t);
    for (; i < number_of_modules; i++){
        if (strcmp(str, nuts_modules[i].name) == 0){ // Module exists
            found = true;
            printf("\rSetup status: %i\r\n", csp_default_setup(nuts_modules[i].module.number));
        }
    }

    if (!found){
        printf("\r\nArgument \"%s\" is not a module,\n\n\r\n"
            "Use one of the following modules:\r\n", str);
        for (i = 0; i < number_of_modules; i++){
            printf("\r%s\r\n", nuts_modules[i].name);
        }
    }
}

void printDiag(portCHAR cmd) {
    extern uint8_t _executable_start;
    unsigned portBASE_TYPE tasksCount = uxTaskGetNumberOfTasks();
    signed char *buffer = pvPortMalloc(tasksCount * 50);

```



```

printf("\rExecutable data starts at %p\r\n", &__executable_start);
printf("\rNumber of tasks is %lu\r\n", tasksCount);
//printf("Number of open files is %u\r\n", io_open_files());

if( buffer != NULL ) {
    vTaskList(buffer);
    printf("%s\n", buffer);
    vPortFree(buffer);
}
}

void taskCmd(const char* pcCommandString){
    printDiag(' ');
}

void cspPrintCommands(const char* args){
    uint16_t i = 0;
    uint16_t tabSz = sizeof(prvCmdTable)/sizeof(CSP_cli_command_definition_t);
    printf("\n\r");
    for(; i < tabSz; i++){
        printf("%s\t- %s\r\n", prvCmdTable[i].pcCommand, prvCmdTable[i].pcHelpString);
    }
}

void cspExit(const char* args){
    csp_test_dir = false;
    //delete_default_listening_task();
    printf("\r\n-----Leaving CSP-----\r\n");
}

int csp_run_command(char* command, char* args){
    uint16_t i = 0;
    uint16_t no_of_commands = sizeof(prvCmdTable)/sizeof(CSP_cli_command_definition_t);
    for(i = 0; i < no_of_commands; i++){
        if(!strcmp(command, prvCmdTable[i].pcCommand)){
            prvCmdTable[i].pxCommandInterpreter(args);
            return 1;
        }
    }
    return -1;
}

```

---

## C.2 CSP example code

### C.2.1 csp\_example.h

---

```

/*
 * csp_example.h
 *
 * Created: 12-Nov-14 12:06:02 PM
 * Author: Erlend
 */

#ifndef CSP_EXAMPLE_H_
#define CSP_EXAMPLE_H_

#include "portmacro.h"

/*
 * Simple example setup
 */
void example_setup_of_listener(portCHAR * listen_port);

/* Simple example of how to send a packet using CSP
 * Function is called from the CSP CLI directory; send

```



```

void example_default_listen_task(void *ptr){
    printf("\rDefault listener task started!\r\n");
    csp_conn_t *conn;
    uint8_t listen_socket = *(uint8_t*)ptr;

    csp_socket_t *sock = create_listen_socket(listen_socket);
    if (sock == NULL) {
        printf("\rError creating/binding port/socket.\r\n");
        vTaskDelete(NULL); //Delete myself
    }
    while(true) {
        conn = csp_accept(sock, CSP_ACCEPT.TIMEOUT);

        // Listener loop (waits for packets)
        while(true){
            csp_packet_t * packet = csp_read(conn, CSP_READ.TIMEOUT);
            if (packet == NULL){
                csp_close(conn);
                break;
            }

            nuts_module_t from_module;

            if (get_nuts_module(packet->id.src, &from_module)
                != SUCCESS){
                from_module.csp_address = packet->id.src;
            }

            printf("\rReceived packet from [%s,%i] at port [%i]: "\
                "[%s]\r\n", from_module.name, packet->id.sport,
                packet->id.dport, packet->data);

        }

        if (conn != NULL)
            csp_close(conn);
    }
}

/*Listen example task*/
void example_listen_task(void *ptr) {
    printf("\rListener task started!\r\n");
    csp_conn_t *conn;
    uint8_t listen_socket = *(uint8_t*)ptr;

    csp_socket_t *sock = create_listen_socket(listen_socket);
    if (sock == NULL) {
        printf("\rError creating/binding port/socket.\r\n");
        vTaskDelete(NULL); //Delete myself
    }

    while(true) {
        conn = csp_accept(sock, CSP_ACCEPT.TIMEOUT);

        // Listener loop (waits for packets)
        while(true){
            csp_packet_t * packet = csp_read(conn, CSP_READ.TIMEOUT);
            if (packet == NULL){
                csp_close(conn);
                break;
            }

            nuts_module_t to_module;

            if (get_nuts_module(packet->id.src, &to_module) != SUCCESS){
                printf("\rReceived packet from [%i,%i]: [%s]\r\n"
                    , packet->id.src, packet->id.sport, packet->data);
                printf("\rError, unable to respond to message because "\
                    "I didn't find \rthe module with csp address: %i from"\
                    "the module list\r\n", packet->id.src);
                break;
            }

            printf("\rReceived packet from [%s,%i] at port [%i]: "\

```

```

        "[%s]\r\n", to_module.name, packet->id.sport,
        packet->id.dport, packet->data);

    module_socket_t to_socket = {to_module, to_module.default_listen_port};
    nuts_module_t from_module = nuts_modules[this_module];
    module_socket_t from_socket = {from_module, 25};

    char *respond_data = "I received a packet from you!";

    csp_transmit(DATA, &to_socket, &from_socket, respond_data,
        strlen(respond_data), NO_COMMAND, 2);
    csp_buffer_free(packet);
}

if (conn != NULL)
    csp_close(conn);
}

void example_loop_test_listening_task(void *ptr){
    printf("\rListener task started!\r\n");
    csp_conn_t *conn;
    uint8_t listen_socket = *(uint8_t*)ptr;

    csp_socket_t *sock = create_listen_socket(listen_socket);
    if (sock == NULL) {
        printf("\rError creating/binding port/socket.\r\n");
        vTaskDelete(NULL); //Delete myself
    }

    while(true) {
        conn = csp_accept(sock, CSP_ACCEPT.TIMEOUT);

        // Listener loop (waits for packets)
        while(true){
            csp_packet_t * packet = csp_read(conn, CSP_READ.TIMEOUT);
            if (packet == NULL){
                csp_close(conn);
                break;
            }

            //toggle led for debugging
            int i = 0;
            for (; i < 2; i++){
                gpio_toggle_pin(AVR32_PIN_PB03);
                vTaskDelay(100);
            }

            nuts_module_t next_module;

            uint8_t next_port;

            switch (this_module){
                case ONBOARD_COMPUTER: //forward to Radio
                    next_module = nuts_modules[RADIO];
                    next_port = RADIO_LOOP_TEST_LISTEN_PORT;

                    break;
                case RADIO: //forward to camera
                    next_module = nuts_modules[CAMERA];
                    next_port = CAMERA_LOOP_TEST_LISTEN_PORT;
                    break;
                case CAMERA: //Forward to obc
                    next_module = nuts_modules[ONBOARD_COMPUTER];
                    next_port = OBC_LOOP_TEST_LISTEN_PORT;
                    break;
                default:
                    printf("\rThis module is not set in module list\r\n");
            }

            module_socket_t next_socket = {next_module, next_port};
            printf("\r\nPackets received = %i\r\n", ++packet_counter_success);

            nuts_module_t from_module = nuts_modules[this_module];
            module_socket_t from_socket = {from_module, 10};

```

```

        char *respond_data = "abcdefghijklmnopqrstuvwxyz0123456789!";

        csp_transmit(DATA, &next_socket, &from_socket, respond_data,
            strlen(respond_data), NO_COMMAND, 2);
        csp_buffer_free(packet);
    }

    if (conn != NULL)
        csp_close(conn);
}

void example_send_simple_message(portCHAR * str){
    int i = 0;

    char *rest = str;

    char *message = strsep(&rest, "@");
    uint8_t address = atoi(strsep(&rest, ":"));
    uint8_t port = atoi(rest);
    printf("\rAddress: %i:%i, Message: %s\r\n", address, port, message);

    nuts_module_t to_module;
    if (address == 2){
        to_module = nuts_modules[ONBOARD_COMPUTER];
    }
    else if (address == 4) to_module = nuts_modules[RADIO];
    else if (address == 6) to_module = nuts_modules[CAMERA];
    else {
        to_module.csp_address = address;
        to_module.i2c_address = address;
    }

    module_socket_t to_socket = {to_module, port};
    module_socket_t from_socket = {nuts_modules[this_module], 25};

    if (strcmp(message, "ping") == 0) message = ping_packet;
    else if (strcmp(message, "big") == 0) message = big_packet;
    else if (strcmp(message, "huge") == 0) message = huge_packet;

    csp_transmit(DATA, &to_socket, &from_socket, message, strlen(message), NO_COMMAND, 2);
}

void example_start_loop_test(){
    uint8_t loop_test_listen_port;
    switch (this_module){
        case ONBOARD_COMPUTER:
            loop_test_listen_port = OBC_LOOP_TEST_LISTEN_PORT;
            break;
        case RADIO:
            loop_test_listen_port = RADIO_LOOP_TEST_LISTEN_PORT;
            break;
        case CAMERA:
            loop_test_listen_port = CAMERA_LOOP_TEST_LISTEN_PORT;
            break;
        default:
            loop_test_listen_port = 25;
    }
    int status = 0;
    if (status = (csp_create_listener(loop_test_listen_port,
        &example_loop_test_listening_task)) != SUCCESS){
        printf("\rError creating listener for loop tests: status: %i\r\n", status);
    }
    packet_counter_success = 0;

    if (this_module == ONBOARD_COMPUTER){
        module_socket_t to_socket = {nuts_modules[RADIO], RADIO_LOOP_TEST_LISTEN_PORT};
        module_socket_t from_socket = {nuts_modules[this_module], OBC_LOOP_TEST_LISTEN_PORT};
        char *data = "start_loop_test";
        csp_transmit(DATA, &to_socket, &from_socket, data, strlen(data), NO_COMMAND, 2);
    }
}

```

```
}
```

---

## C.3 CSP controller code

### C.3.1 csp\_controller.h

---

```
/*
 * csp_controller.h
 *
 * Created: 10-Nov-14 4:28:24 PM
 * Author: Erlend
 */

#ifndef CSP_CONTROLLER_H_
#define CSP_CONTROLLER_H_

#include "projdefs.h"
#include "conf_csp.h"
#include "portmacro.h"
#include "csplib/csp_util.h"
#include "csplib/include/csp/csp.h"
// #include "bit_patterns.h"

/*****
 * DEFINES
 *****/

/*Module definitions*/
#define ONBOARD_COMPUTER 0
#define RADIO 1
#define CAMERA 2

// #####
// THE DEFINITION OF WHICH CUBESAT MODULE THIS IS !!!
// #define MODULE OBC
// #####

// I2C addresses
#define OBC_I2C_ADDR 0x02
#define RADIO_I2C_ADDR 0x04
#define CAMERA_I2C_ADDR 0x06

// CSP addresses
#define OBC_CSP_ADDR OBC_I2C_ADDR
#define RADIO_CSP_ADDR RADIO_I2C_ADDR
#define CAMERA_CSP_ADDR CAMERA_I2C_ADDR

// Destination/listener port examples MAX 6bit
#define OBC_DEFAULT_LISTEN_PORT 0x10
#define RADIO_DEFAULT_LISTEN_PORT 0x11
#define CAMERA_DEFAULT_LISTEN_PORT 0x12

#define OBC_LOOP_TEST_LISTEN_PORT 0x17
#define RADIO_LOOP_TEST_LISTEN_PORT 0x18
#define CAMERA_LOOP_TEST_LISTEN_PORT 0x19

// Other port examples that can be changed as desired
#define OBC_FILE_SYSTEM_LISTEN_PORT 0x13
#define OBC_DETUMBLE_LISTEN_PORT 0x14
#define RADIO_TO_GROUND_LISTEN_PORT 0x15
#define CAMERA_COMMAND_LISTEN_PORT 0x16

// CSP specifics
#define MAX_ADDR 31
#define MAX_PORT 63
```

```

// CSP timeout constants
#define CSP_SEND_TIMEOUT 1000
#define CSP_ACCEPT_TIMEOUT 20000
#define CSP_READ_TIMEOUT 0

// Command settings
#define ASCII_LENQ 0x05 // Used to indicate command packet
#define ASCII_EOT 0x04 // Sent after last byte of split packet
#define MAX_COMMANDS 256 // Maximum number of commands
#define COMMAND_SIZE 2 // Size of command in packet

// Split packet
#define MAX_PACKET_SIZE 60 // Size of data to send in each packet. max 61
#define MAX_PACKET_SPLIT_SIZE 20*MAX_PACKET_SIZE // Maximum number of packets to split/combine
#define PACKET_NUM_SIZE 2 // Bytes that define number of split packets
#define PACKET_NUM_CMD_SIZE PACKET_NUM_SIZE+COMMAND_SIZE // Size of packet number and command bytes

// Delay values
#define SPLIT_SEND_DELAY 100 // Time to wait before sending next part of split packet
#define SEND_MANY_PACKETS_DELAY 100 // Interval between sending of packets in "send many packets"-test

// Errors - add more
#define SUCCESS 0
#define ADDRESS_OUT_OF_RANGE -4
#define PORT_NUMBER_OUT_OF_RANGE -5
#define UNABLE_TO_INITIALIZE_CSP -6
#define UNABLE_TO_CREATE_LISTENER -7
#define NO_SUCH_MODULE -8

/*****
 * TYPEDEFS
 *****/
typedef enum {
    TO_SLAVE = 1,
    TO_MASTER = 0
} transfer_direction;

typedef struct {
    uint8_t module_number;
    uint8_t csp_address;
    uint8_t i2c_address;
    uint8_t default_listen_port;
    char name[10];
} nuts_module_t;

typedef struct {
    nuts_module_t module;
    uint8_t port;
} module_socket_t;

typedef enum {
    SRC = 0,
    DEST = 1
} src_dest_t;

typedef enum {
    COMMAND = 0,
    DATA = 1,
    COMMAND_PLUS_DATA = 2
} packet_type_t;

typedef enum {
    CMD_DEFAULT = 0,
    CMD_SEND_BACK = 1,
    CMD_BOUNCE = 2,
    CMD_RETURN = 3,
    CMD_RETURNED = 4,
    CMD_SPLIT_START = 5,
    CMD_SPLIT_STOP = 6,
    CMD_CIRCLE = 7,
    CMD_SPLIT = 8,
    CMD_PING_REPLY = 9,
    CMD_PATTERN = 10,
    NO_COMMAND = 98,
    CMD_DO_NOTHING = 99
} command_t;

```

```

typedef enum {
    BIT_PAT_1_IN_7 = 0,
    BIT_PAT_ALL_ONES,
    BIT_PAT_ALL_ZEROS,
    BIT_PAT_3_IN_24,
    BIT_PAT_QRSS,
    BIT_PAT_ALT
} bit_pattern_t;

/*Examples of modules, add and remove as wanted*/
static const nuts_module_t nuts_modules [] = {
    {OBC, OBC_CSP_ADDR, OBC_I2C_ADDR, OBC_DEFAULT_LISTEN_PORT, "OBC"},
    {RADIO, RADIO_CSP_ADDR, RADIO_I2C_ADDR, RADIO_DEFAULT_LISTEN_PORT, "RADIO"},
    {CAMERA, CAMERA_CSP_ADDR, CAMERA_I2C_ADDR, CAMERA_DEFAULT_LISTEN_PORT, "CAMERA"}
};

cubesat_module this_module;

/*****
 * Methods
 *****/

/* Initializes CSP for a given address. Cannot be changed when initialized.
 *
 * \param: the defined module number
 * \return: 0 on success, negative numbers on error
 */
int8_t csp_default_setup(uint8_t module_number);

/* Populates a nuts_module_t struct from the defined module structs in the
 * nuts_modules list
 *
 * \param: csp_address: address of the module you want to copy
 *         *module_ptr: the module you want to populate
 * \return 0 on success, negative on error
 *
 * \NOTE: Unnecesary, as module = nuts_modules[OBC/RADIO/CAMERA]; will work to..
 */
int8_t get_nuts_module(uint8_t csp_address, nuts_module_t *module_ptr);

/* Creates FreeRTOS task to be used as a socket listener for CSP
 *
 * \param: listen_port: the port number for the socket
 *         *listener_ptr: pointer to a function that will be ran by the FreeRTOS task
 * \return: 0 on success, negative on error
 */
int8_t csp_create_listener(uint8_t listen_port, void (*listener_ptr)(void *arg_ptr));

/* Creates a socket to be used in the listener
 *
 * \param: port: the port number for the socket
 * \return: a pointer to the created socket
 */
csp_socket_t * create_listen_socket(uint8_t port);

/* transmits data over CSP
 *
 * \param: type: DATA/COMMAND/DATA.AND.COMMAND - Specifying what data is being sent
 *         *dest: pointer to destination socket of type module_socket_t
 *         *source: pointer to source socket of type module_socket_t
 *         *data: pointer to the data being sent
 *         size: size of data
 *         command: the command to be sent with the data if type is set to COMMAND or
 *                  DATA and COMMAND
 *         priority: the priority of the transmission (0-3, 0 being highest priority)
 * \return: 0 on success, negative on error
 */
int8_t csp_transmit(packet_type_t type, module_socket_t *dest, module_socket_t *source,
    char *data, int size, command_t command, csp_prio_t priority);

/* Method used to split an oversized packet into smaller packets that are sent individually.

```



```

* Uses the CMD_SPLIT_START command in the first packet, and CMD_SPLIT_STOP command in
* the last packet, which can be used to regenerate the data on the receiving end.
* The method is called automatically if the size of data
* in the csp_transmit call is above 60 bytes
*
*\param: size: size of data
*       *i2c_address: address of receiver
*       *dest: destination port
*       *src: source port
*       *data: data to be sent
*       num_packets: number of packets to be split into
*       priority: the priority of the csp transmission
*\return: 0 on success, negative on error
*/
int8_t csp_split_packet(int size, uint8_t *i2c_addr, uint8_t *dest, uint8_t *src,
    char *data, int num_packets, csp_prio_t priority);

/* Sending a csp packet containing data only
*
*\param: *i2c_address: address of receiver
*       *dest: destination port
*       *src: source port
*       *data: data to be sent
*       size: size of data
*       priority: the priority of the csp transmission
*\return: 0 on success, negative on error
*/
int8_t csp_send_data(uint8_t *i2c_addr, uint8_t *dest_port, uint8_t *src_port,
    char *data, uint8_t size, csp_prio_t priority);

/* Sending a csp packet containing command only
*
*\param: *i2c_address: address of receiver
*       *dest_port: destination port
*       *src_port: source port
*       command: the command
*       priority: the priority of the csp transmission
*\return: 0 on success, negative on error
*/
int8_t csp_send_command(uint8_t *i2c_addr, uint8_t *dest_port, uint8_t *src_port,
    command_t command, csp_prio_t priority);

/* Sending a packet containing both a command and data. command is placed in
* the beginning of the packet
*
*\param: *i2c_address: address of receiver
*       *dest_port: destination port
*       *src_port: source port
*       *data: data to be sent
*       size: size of data
*       command: command to be sent with the data
*       priority: the priority of the csp transmission
*\return: 0 on success, negative on error
*/
int8_t csp_send_command_and_data(uint8_t *i2c_addr, uint8_t *dest_port, uint8_t *src_port,
    char *data, uint8_t size, command_t command, csp_prio_t priority);

/* Calls the csp sending function from the CSP library
*
*\param: *i2c_address: address of receiver
*       *dest: destination port
*       *src: source port
*       csp_flags: option to set flags for encryption of data/etc
*       *packet: csp_packet_t struct containing the packet
*       priority: the priority of the csp transmission
*\return: 0 on success, negative on error
*/
int8_t csp_send_packet(uint8_t *i2c_addr, uint8_t *dest_port, uint8_t *src_port,
    uint8_t csp_flags, csp_packet_t *packet, csp_prio_t priority);

void * csp_buffer_get_wrapper(size_t size);
#endif /* CSP_CONTROLLER_H */

```

---

## C.3.2 csp\_controller.c

---

```
/*
 * csp_controller.c
 *
 * Created: 10-Nov-14 4:42:04 PM
 * Author: Code was written by EIT Romteknologi gr.5, and edited by Erlend Riis Jahren
 */

#include "csp_controller.h"
#include <stdio.h>
#include <string.h>
#include <asm.h>
#include "FreeRTOS.h"
#include "portmacro.h" // For tick length information / timers
#include "task.h"
#include "status_codes.h"
#include "csplib/csp_util.h"
#include "csplib/i2c-csp-glue.h"
#include "csplib/include/csp/csp_debug.h"
#include "cli/cli.h"
#include "cli/cli_commands/csp_tests.h"
#include "csp_example.h"

// Global taskhandles for all the listening tasks
static xTaskHandle xCSPListenPort[MAX_PORT - 1];

/*CSP initialization*/
uint8_t csp_initialized = false;
uint8_t csp_usage_init_initialized = false;
uint8_t csp_default_setup(uint8_t module_number){
    if (!csp_initialized){
        //initialize all listen task handlers
        int i = 0;
        for (; i < MAX_PORT; i++){
            xCSPListenPort[i] = NULL;
        }
        this_module = module_number;
        if (csp_usage_init(CSP_NR_OF_BUFFERED_PACKETS, nuts_modules[this_module].csp_address,
            configTSK_ROUTINGTASK_STACK_SIZE) != 0)
            return UNABLE_TO_INITIALIZE_CSP;

        csp_usage_init_initialized = true;

        csp_initialized = true;

    }
    else printf("\rCSP already initialized\r\n");

    printModuleInfo();
    return SUCCESS;
}

int8_t get_nuts_module(uint8_t csp_address, nuts_module_t *module_ptr){
    int i = 0;
    for (; i < sizeof(nuts_modules)/sizeof(nuts_module_t); i++){
        if (nuts_modules[i].csp_address == csp_address){
            *module_ptr = nuts_modules[i];
            return SUCCESS;
        }
    }
    return NO_SUCH_MODULE;
}

/*Listener setup*/
int8_t csp_create_listener(uint8_t listen_port, void (*listener_ptr)(void *arg_ptr)){
    int status = SUCCESS;
    xCSPListenPort[listen_port] = NULL;
    char task_name[16];
    snprintf(task_name, 16, "listen[%i,%i]", nuts_modules[this_module].csp_address, listen_port);
```

```

    if (status = xTaskCreate(listener_ptr, task_name, configTSK_DEBUGTASK_STACK_SIZE,
        (void*)&listen_port, configTSK_DEBUGTASK_PRIORITY, &xCSPListenPort[listen_port])
        != pdPASS){
        printf("\rERROR creating listener task: %i\r\n", status);
    }
    return status;
}

/*Set up a socket for your listener*/
csp_socket_t * create_listen_socket(uint8_t port) {
    //create the socket
    csp_socket_t *listen_socket = csp_socket(CSP_SO_NONE);
    int status;
    if (listen_socket == NULL){
        printf("\rError creating socket\r\n");
        return NULL;
    }

    //Bind socket to port
    if ((status = csp_bind(listen_socket, port)) != CSP_ERR_NONE){
        printf("\rError binding socket, status: %i\r\n", status);
        return NULL;
    }

    //Activate listening for socket
    if ((status = csp_listen(listen_socket, BACKLOG_CON_QUEUE)) != CSP_ERR_NONE){
        printf("\rError listening to socket, status: %i\r\n", status);
        return NULL;
    }

    return listen_socket;
}

/*
*/
int8_t csp_transmit(packet_type_t type, module_socket_t *dest, module_socket_t *source,
    char *data, int size, command_t command, csp_prio_t priority) {
    int retval[3] = {1};
    static uint16_t num_packets;

    // Use different approaches for the different packet types
    switch(type) {
        case COMMAND:
            retval[0] = csp_send_command(&dest->module.i2c_address, &dest->port,
                &source->port, command, priority);
            break;
        case DATA:
            // Check if packet has to be split
            if (size > MAX_PACKET_SIZE) {
                if (!(size % (MAX_PACKET_SIZE - COMMAND_SIZE + 1))) {
                    // data size fits exactly (+EOT)
                    num_packets = (size + 1) / (MAX_PACKET_SIZE - COMMAND_SIZE);
                }
                else {
                    num_packets = (size + 1) / (MAX_PACKET_SIZE - COMMAND_SIZE) + 1;
                }
            }
            //if (verbose)
            //printf("Splitting into '%i' packets.\r\n", num_packets);

            char num_char[2], *ptr_num_char;
            num_char[0] = num_packets & 0xFF;
            num_char[1] = (num_packets >> 8);
            ptr_num_char = num_char;

            retval[0] = csp_send_command_and_data(&dest->module.i2c_address,
                &dest->port, &source->port, ptr_num_char, PACKET_NUM_SIZE,
                CMD_SPLIT_START, priority);
            retval[1] = csp_split_packet(size, &dest->module.i2c_address, &dest->port,
                &source->port, data, num_packets, priority);
            retval[2] = csp_send_command(&dest->module.i2c_address, &dest->port,
                &source->port, CMD_SPLIT_STOP, priority);
    }
}

```

```

else {
    retval[0] = csp_send_data(&dest->module.i2c_address, &dest->port,
        &source->port, data, size, priority);
}
break;
case COMMAND_PLUS_DATA:
    // Check if packet has to be split
    if (size > MAX_PACKET_SIZE) {

        if (!(size % (MAX_PACKET_SIZE-COMMAND_SIZE+COMMAND_SIZE+1))) {
            // data size + command bytes fits exactly (+EOT)
            num_packets = (size+COMMAND_SIZE+1)/(MAX_PACKET_SIZE-COMMAND_SIZE);
        }
        else {
            num_packets = (size+COMMAND_SIZE+1)/(MAX_PACKET_SIZE-COMMAND_SIZE)+1;
        }

        char num_char[3];
        num_char[0] = num_packets & 0xFF;
        num_char[1] = (num_packets >> 8);
        num_char[2] = ASCII_ENQ;
        num_char[3] = command;

        retval[0] = csp_send_command_and_data(&dest->module.i2c_address,
            &dest->port, &source->port, &num_char, PACKET_NUM_SIZE+COMMAND_SIZE,
            CMD_SPLIT_START, priority);
        retval[1] = csp_split_packet(size, &dest->module.i2c_address,
            &dest->port, &source->port, data, num_packets, priority);
        retval[2] = csp_send_command(&dest->module.i2c_address,
            &dest->port, &source->port, CMD_SPLIT_STOP, priority);
    }
    else {
        retval[0] = csp_send_command_and_data(&dest->module.i2c_address,
            &dest->port, &source->port, data, size, command, priority);
    }
    break;
}
// Return the status code
if (retval[1] == 1 && retval[2] == 1) {
    return retval[0];
}
else {
    for (int i=0; i<3; i++) {
        if (retval[i] != 0) {
            return retval[i]; // only one error returned if many
        }
    }
    return STATUS_OK;
}
}

int8_t csp_split_packet(int size, uint8_t *i2c_addr, uint8_t *dest,
    uint8_t *src, char *data, int num_packets, csp_prio_t priority) {
    char small_packet[MAX_PACKET_SIZE];
    char *ptr_packet = small_packet;
    int retval;
    int sent_packets = 0;

    for (int i=0; i<num_packets; i++) {

        // Last packet
        if (sent_packets == num_packets-1) {
            int last_size = size-MAX_PACKET_SIZE*i;

            // Last data to be sent. Add EOT to end.
            if (last_size > 0) {
                memcpy(ptr_packet, data+i*MAX_PACKET_SIZE, last_size);
                small_packet[last_size] = ASCII_EOT;
                last_size++;
            }
            // Only EOT byte to be sent. Data size must be at least 2, so packet becomes 4.
            else if (last_size == 0) {
                small_packet[0] = ASCII_EOT;
                small_packet[1] = ASCII_EOT;
                last_size += 2;
            }
        }
    }
}

```

```

        ret_val = csp_send_command_and_data(i2c_addr, dest, src, ptr_packet,
                                            last_size, CMD_SPLIT, priority);
    }
    // Send the current portion of the data
    else {
        ret_val = csp_send_command_and_data(i2c_addr, dest, src,
                                            data+i*MAX_PACKET_SIZE, MAX_PACKET_SIZE, CMD_SPLIT, priority);
    }
    sent_packets++;

    if (ret_val != STATUS_OK) {
        break;
    }
    // Some delay to avoid congestion
    vTaskDelay(SPLIT_SEND_DELAY);
}
return ret_val;
}

int8_t csp_send_data(uint8_t *i2c_addr, uint8_t *dest_port, uint8_t *src_port,
                    char *data, uint8_t size, csp_prio_t priority) {
    csp_packet_t *packet = csp_buffer_get_wrapper(sizeof(csp_packet_t) + size);

    // Copy the data to packet and send.
    if (packet != NULL) {
        packet->length = size;
        memcpy(packet->data, data, size);

        if (size == 0)
            printf("Sending packet with length 0. No connection will be made on receiving module!\r\n");

        if (csp_send_packet(i2c_addr, dest_port, src_port, CSP_O_NONE, packet, priority) == ERR_IO_ERROR) {
            printf("Data not sent.\r\n");

            return ERR_IO_ERROR;
        }
        else {
            // if (verbose) {
            //     printf("Data '%.*s' sent | ", packet->length, packet->data);
            //     printf("With length: %i\r\n", packet->length);
            // }
            packet = NULL;

            return STATUS_OK;
        }
    }
    else {
        return ERR_NO_MEMORY;
    }
}

int8_t csp_send_command(uint8_t *i2c_addr, uint8_t *dest_port, uint8_t *src_port,
                        command_t command, csp_prio_t priority) {
    // Size is the packet struct itself plus 4 bytes
    // (two extra due to small packets being rejected).
    csp_packet_t *packet = csp_buffer_get_wrapper(sizeof(csp_packet_t) + COMMAND_SIZE + 2);

    if (packet != NULL) {
        // Get command into packet data
        if (command > 0 && command < MAX_COMMANDS) {
            packet->length = 4; // Packet has to be at least this size in order
                               // to be detected on receiver. Why?
            packet->data[0] = ASCII_ENQ;
            packet->data[1] = command;
        }
        else {
            return ERR_BAD_DATA;
        }
        // Send packet.
        if (csp_send_packet(i2c_addr, dest_port, src_port, CSP_O_NONE, packet, priority)
            == ERR_IO_ERROR) {
            printf("Command not sent.\r\n");

            return ERR_IO_ERROR;
        }
    }
}

```

```

        else {
            packet = NULL;
            return STATUS_OK;
        }
    }
    else {
        return ERR_NO_MEMORY;
    }
}

int8_t csp_send_command_and_data(uint8_t *i2c_addr, uint8_t *dest_port,
    uint8_t *src_port, char *data, uint8_t size, command_t command, csp_prio_t priority) {
    csp_packet_t *packet = csp_buffer_get_wrapper(sizeof(csp_packet_t) +
        size + COMMAND_SIZE);

    if (packet != NULL) {
        // Get command into packet data
        if (command >= 0 && command < MAX_COMMANDS) {
            packet->data[0] = ASCII_ENQ;
            packet->data[1] = command;
        }
        else {
            return ERR_BAD_DATA;
        }
        // Copy the data to be sent
        packet->length = size + COMMAND_SIZE;
        memcpy(&packet->data[2], data, size);

        // Send packet. Release buffer if fails
        if (csp_send_packet(i2c_addr, dest_port, src_port, CSP_O_NONE,
            packet, priority) == ERR_IO_ERROR) {
            csp_buffer_free(packet);
            printf("Command+data not sent.\r\n");

            return ERR_IO_ERROR;
        }
        else {
            packet = NULL;
            return STATUS_OK;
        }
    }
    return ERR_IO_ERROR;
}

int8_t csp_send_packet(uint8_t *i2c_addr, uint8_t *dest_port, uint8_t *src_port,
    uint8_t csp_flags, csp_packet_t *packet, csp_prio_t priority) {
    // Send packet
    int err;
    if ((err = csp_sendto(priority, *i2c_addr, *dest_port, *src_port, csp_flags,
        packet, CSP_SEND_TIMEOUT)) == ERR_IO_ERROR) {
        csp_buffer_free(packet);
        return ERR_IO_ERROR;
    }
    else {
        packet = NULL;
        return STATUS_OK;
    }
}

void * csp_buffer_get_wrapper(size_t size){
    csp_packet_t *packet = csp_buffer_get(size);
    packet->id.ext = 0;
    packet->length = 0;
    for(int i = 0; i < CSP_PADDING_BYTES; i++){
        packet->padding[i] = '\0';
    }
    return packet;
}

```

---