Lucerne University of
Applied Sciences and Arts

HOCHSCHULE
LUZERN

Technik & Architektur
FH Zentralschweiz

MSE | MASTER OF SCIENCE
IN ENGINEERING

UAV Serial Switch
User Manual

Stefanie Schmidiger

MASTER OF SCIENCE IN ENGINEERING
Vertiefungsmodul I

Advisor: Prof. Erich Styger

Experte: Dr. Christian Vetterli

Horw, 2018

Table of Contents

1 Introduction

The exchange of data between on-board and off-board components is of vital importance with unmanned vehicles. Usually, each sensor or actor is directly connected to a modem (see Figure 1.1)and dynamic data routing data between multiple devices and modems is not possible. When data transmission fails over one transmission technology, an intermediate platform is needed to detect this unreliable connection and quickly switch from one transmission technology to an other to ensure a stable data stream between vehicle and ground station.
The UAV Serial Switch provides a solution to this problem. It is a platform for flexible data routing between devices and modems. It features four RS232 interfaces to connect data generating and processing devices (such as sensors and actors) and four RS232 interfaces to connect modems for data transmission. With jumpers, you either chose an USB connector or the RS232 interface as an input for each of the four serial connections available on device side. An possible use case can bee seen in Figure 1.2
The main functionality of this application is the sending of data packages. The software collects data from its devices, puts them into data packages with header, payload and CRC and sends those data packages out over the configured wireless serial connection. The corresponding second hardware receives the packages on its wireless side, extracts its payload and sends it out on the correct device side.
Acknowledges can be configured for each wireless connection. When a data package is received over

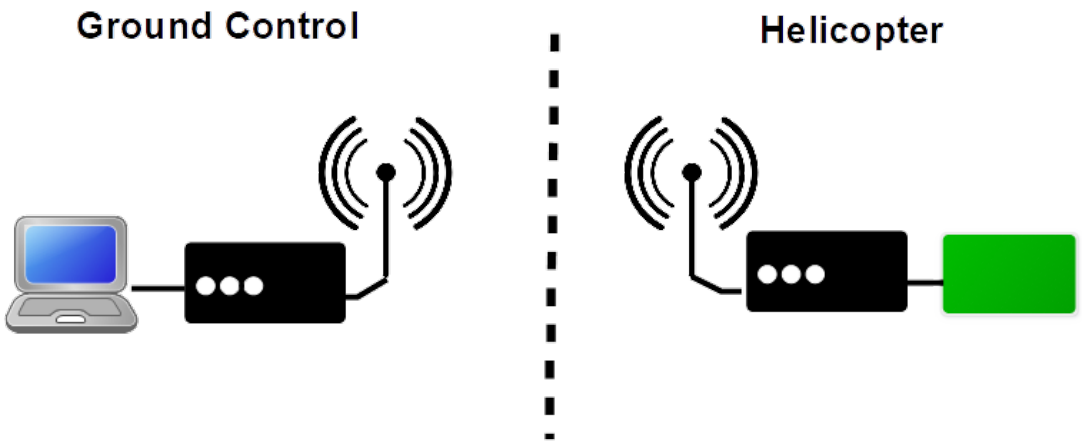

Figure 1.1: Sensors and actors directly connected to a modem

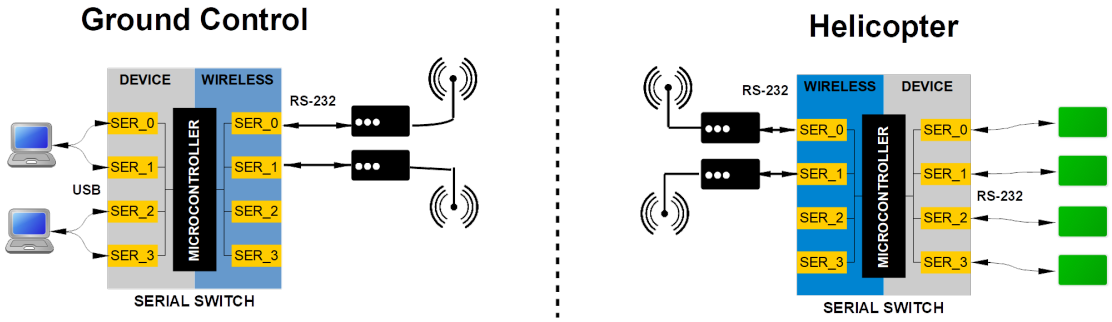

Figure 1.2: UAV Serial Switch as intermediate platform between devices and modems

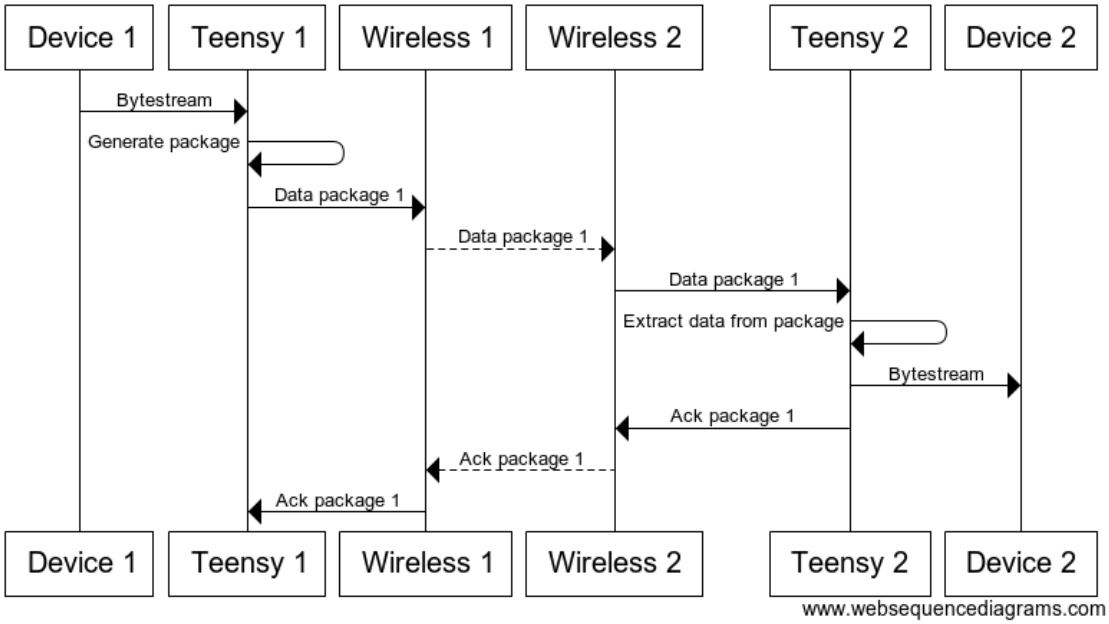

Figure 1.3: Assembling device data into packages

this wireless connection, an acknowledges is generated and returned (see Figure 1.3). If the sender receives no acknowledges within the configured time, it will resend the data package.
The configuration file saved on the SD card leaves you in full control of the functionality of the Serial Switch. Modify it to change data routing between connected devices and modems, baud rate for each RS232 and USB connection and transmission behavior.
For details about hardware configurations, please read Chapter 2 and for details about software configurations, please read Chapter 3.

2 Hardware Configuration

The UAV Serial Switch can be seen in Figure 2.1 and aims to provide the user with maximum flexibility.
It consists of a base board (see Figure 2.2), a Teensy adapter board (see Figure 2.4) and a Teensy 3.5 (see Figure 2.3).
The base board was designed for a Teensy 3.2 but is now used with a Teensy 3.5, hence the adapter board.
Read this chapter to learn more about the connectors and possible hardware configurations accessible to the user.

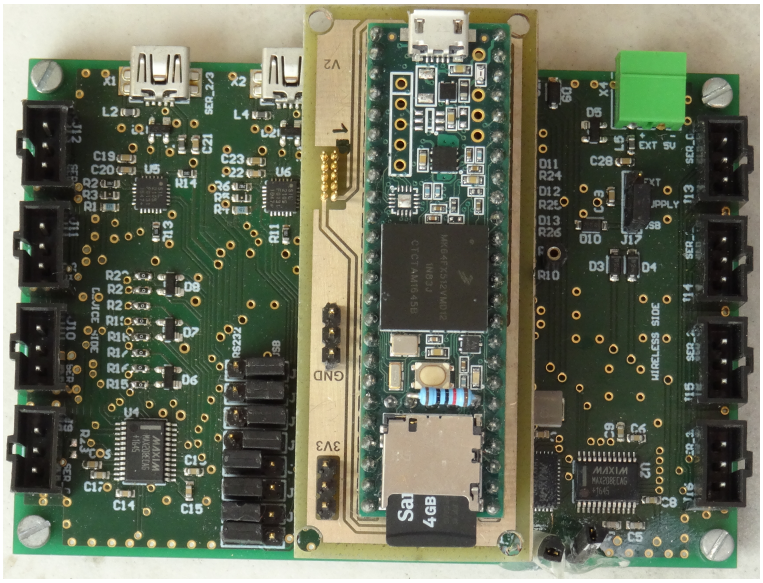

Figure 2.1: UAV Serial Switch

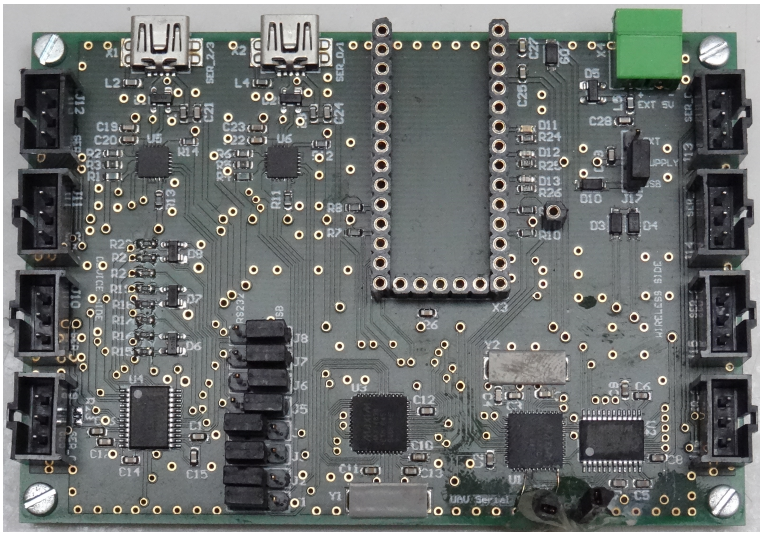

Figure 2.2: Base board

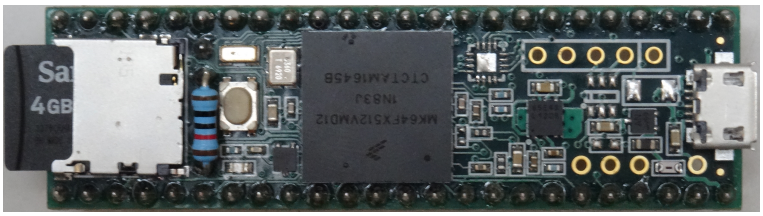

Figure 2.3: Teensy 3.5 development board

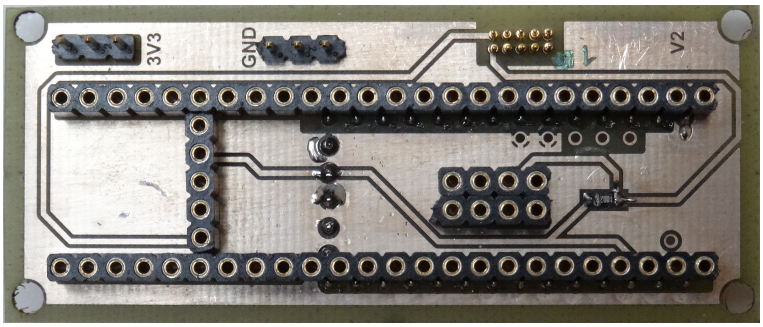

Figure 2.4: Teensy adapter board

2.1 User Interfaces

From now on, the side where data generating and data processing devices can be connected will be referred to as the device side and the side where modems can be connected will be referred to as the wireless side. See Figure 2.5 for the location of the connectors.

2.1.1 RS232 Interface

There are four RS232 interfaces available for both device and wireless side. The sigals available are kept to a bare minimum with only RX, TX and ground lines. See Figure 2.6 for details on pinout of the RS232 interfaces. All RS232 connectors on device side have the same pinout and all RS232 connectors on wireless side have the same pinout.
Connector numbering is also visible in Figure 2.6.

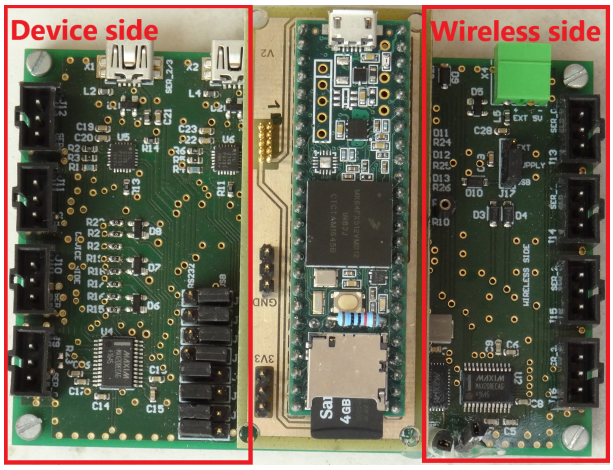

Figure 2.5: Device side and wireless side

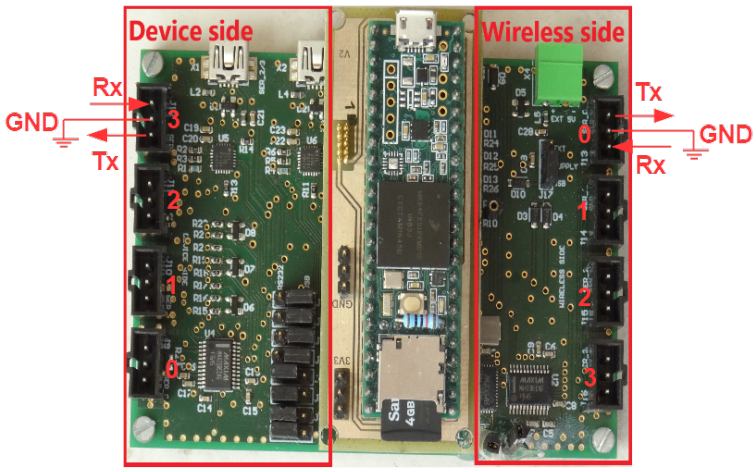

Figure 2.6: Pinout of RS232 user interfaces

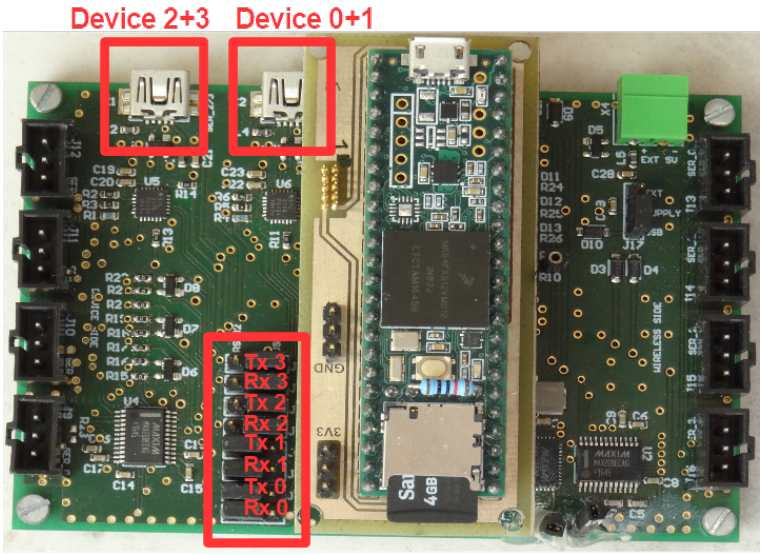

Figure 2.7: USB user interfaces

2.1.2 USB Serial Port

Instead of connecting devices on the RS232 interfaces, you can configure the Serial Switch for USB connected devices.
There are two USB connectors available on the base board. They act as dual USB to serial bridges, so each USB interface can replace two RS232 interfaces (see Figure 2.7). When connecting one USB interface of the base board to a computer, two COM ports will appear to simulate two serial interfaces.
In order to use an USB interface instead of the RS232 connector for a device, the jumpers have to be set accordingly on the base board. See Figure 2.7 to find out which jumper corresponds to which serial interface.
Set the jumpers of the Tx and Rx signals to the left to select the RS232 signal and set it to the right to select the USB interface.

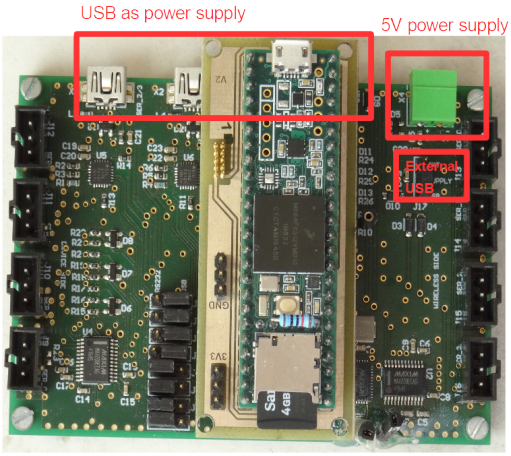

Figure 2.8: Power supply options

2.2 Power Supply

The base board and the adapter board share one power supply. The setup can either be powered by USB from any of the USB connectors or with an external 5V power supply (see Figure 2.8). To select either one of those options, simply set the jumper to the correct position (External or USB, as indicated in Figure 2.8).

2.3 UART Flow Control

Because the RS232 connectors only route the RX and TX signals to the hardware buffer, a workaround has been done to use the CTS flow control signal as well for device connection 0 and 1.
To enable hardware flow control, connect the CTS signals of the RS232 interface to the indicated pin (see Figure 2.9). Careful: the CTS pin is not RS232 level compatible, you need a level shifter because the CTS pin only allows 0V to 5V!

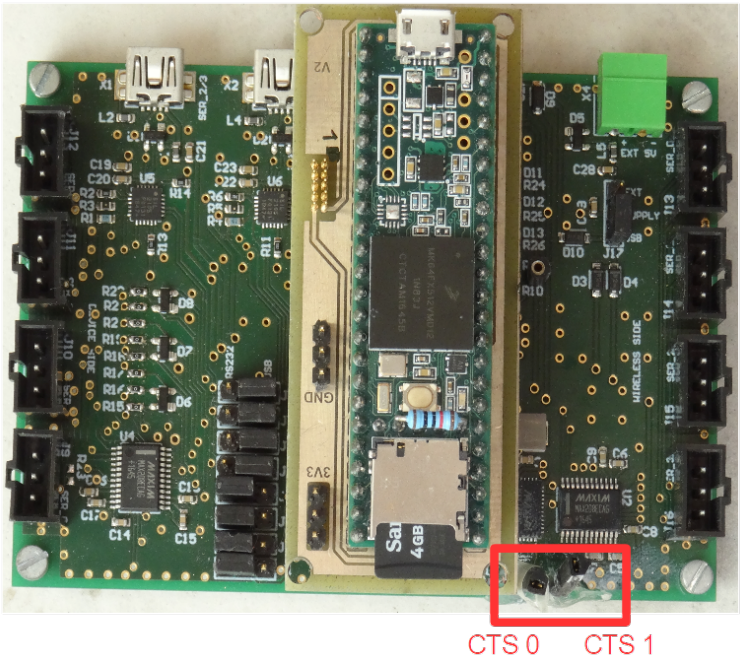

Figure 2.9: Hardware flow control signal

3 Software Configuration

The Teensy 3.5 development board acts as the main micro controller. It collects data from the serial connections on device side, puts them into data packages with header and checksum and sends them out on the configured wireless side. When acknowledges are configured, the software will possibly resend the package if no acknowledge is received. The corresponding second hardware receives packages on its wireless side, extracts the payload and sends it out on its device side.
All communication is bidirectional, so data can be sent and received from all serial interfaces.
The Teensy has an on-board SD card slot with a configuration file on the micro SD card. All software configurations are read from that file upon power up of the Teensy. Possible configuration parameters can be seen in Table 3.1:

Configuration parameter	**Possible values**	**Description**
BAUD_RATES_WIRELESS_CONN	9600, 38400, 57600, 115200	Baud rate to use on wireless side, configurable per wireless connection. Example: 9600, 38400, 57600, 115200 would result in 9600 baud for wireless connection 0, 38400 baud for wireless connection 1 etc.
BAUD_RATES_DEVICE_CONN	9600, 38400, 57600, 115200	Baud rate to use on device side, configurable per cevice connection. Example: 9600, 38400, 57600, 115200 would result in 9600 baud for device connection 0, 38400 baud for device connection 1 etc.
PRIO_WIRELESS_CONN_DEV_X	0, 1, 2, 3, 4	This parameter determines over which wireless connection the data stream of a device will possibly be sent out. 0: this wireless connection will not be used. 1: Highest priority, data will be tried to send out over this connection first. 2: Second highest priority, data will be tried to send out over this connection should transmission over the first priority connection fail. 3: Third highest priority. 4: Lowest priority for data transmission. Example: 0, 2, 1, 0 would result in data being sent out over wireless connection 2 first and only sent out over wireless connection 1 in case of failure. All other wireless connections would not be used. Replace the X in the parameter name with 0, 1, 2 or 3.

SEND_CNT_WIRELESS_ CONN_DEV_X	0...255	Determines how many times a package should tried to be sent out over a wireless connection before moving on to retrying with the next lower priority wireless connection. Example: 0, 5, 4, 0 would result in the package being sent out over wireless connection 1 five times and four times over wireless connection 2. Together with PRIO_WIRELESS_CONN_DEV_X, this parameter determines the number of resends per connection. Replace the X in the parameter name with 0, 1, 2 or 3.
RESEND_DELAY_WIRELESS_ CONN_DEV_X	0...255	Determines how many milliseconds the software should wait for an acknowledge per wireless connection before sending the same package again. Example: 10, 0, 0, 0 would result in the software waiting for an acknowledge for 10ms when having sent a package out via wireless connection 0 before attempting a resend. Together with PRIO_WIRELESS_CONN_DEV_X, this parameter determines the delay of the resend behaviour Replace the X in the parameter name with 0, 1, 2 or 3.
USUAL_PACKET_SIZE_DEVICE_ CONN	0...512	Maximum number of payload bytes per wireless package. 0: unknown payload-> the PACKAGE_GEN_MAX_TIMEOUT parameter always determines the payload size. Example: 128, 0, 128, 128 results in a maximum payload of 128 bytes per package and an unknown maximum payload size for wireless connection 0.
PACKAGE_GEN_MAX_TIMEOUT	0...255	Maximum time (in milliseconds) that the software should wait for a package to fill up before sending it out anyway. Together with USUAL_PACKET_SIZE_DEVICE_CONN, this parameter determines the size of a package. Example: 50, 50, 50, 50 will result in data being sent out after a maximum wait time of 50ms.
DELAY_DISMISS_OLD_PACK_ PER_DEV	0...255	Maximum time (in milliseconds) an old package should be tried to resend while the next package with data from the same device is available for sending. Example: 5, 5, 5, 5 results in a package being discarded 5ms after the next package is available in case it has not been sent successfully until then.

SEND_ACK_PER_WIRELESS_CONN	0, 1	Acknowledges turned on/off for each wireless connection. Example: 1, 1, 0, 0 results in acknowledges being expected and sent over wireless connection 0 and 1 but not over wireless connection 2 and 3.
USE_CTS_PER_WIRELESS_CONN	0, 1	Hardware flow control turned on/off for each wireless connection. Example: 1, 1, 0, 0 results in hardware flow control (CTS) for wireless connection 0 and 1 only. Currently, the hardware only supports CTS being enabled for these two connections as other CTS signals are not accessible to the user.
TEST_HW_LOOPBACK_ONLY	0, 1	This parameter enables local echo. Any data received over any serial connection will be returned over the same connection immediately.
GENERATE_DEBUG_OUTPUT	0, 1	Information about the data throughput and other warnings will be printed out on the serial terminal.
X_TASK_INTERVAL	1 ... 65535	Task interval in milliseconds for each task. Replace X with the name of the task. Use a number greater than 2 to ensure that no task is blocking.

Table 3.1: Software configuration parameters

For task intervals, use a value greater than 2 milliseconds for SPI Handler, Package Handler and Network Handler. These three tasks provide the main functionality of the software as they do all the data processing (see documentation for details). Their task interval determines how frequently data is pulled from the devices and assembled to packages respectively extracted from packages and pushed out to the devices. The shell interval can be set to 50 milliseconds or even higher as it only determines how frequently the command interface is polled.
The configuration file needs to be named serialSwitch_Config.ini
A sample configuration file can be seen below.

```
1
2  ;================================================================================
3  [BaudRateConfiguration]
4  ;
5  ;
6  ; BAUD_RATES_WIRELESS_CONN
7  ; Configuration of baud rates on wireless side from 0 to 3.
8  ; Regarding the supported baud rates see implementation of hwBufIfConfigureBaudRate in
       hwBufferInterface.cpp
9  BAUD_RATES_WIRELESS_CONN = 57600, 38400, 57600, 57600
10 ;
11 ;
12 ; BAUD_RATES_DEVICE_CONN
13 ; Configuration of baud rates on wireless side from 0 to 3.
14 ; Regarding the supported baud rates see implementation of hwBufIfConfigureBaudRate in
       hwBufferInterface.cpp
15 BAUD_RATES_DEVICE_CONN = 57600, 57600, 38400, 38400
16 ;
17 ;
18 ;================================================================================
19 [ConnectionConfiguration]
```

```
;
;
; PRIO_WIRELESS_CONN_DEV_X
; Priority of the different wireless connections from the viewpoint of a single device.
; 0: Wireless connection is not used; 1: Highes priority; 2: Second priority, ..
PRIO_WIRELESS_CONN_DEV_0 = 1, 0, 0, 0
PRIO_WIRELESS_CONN_DEV_1 = 0, 1, 0, 0
PRIO_WIRELESS_CONN_DEV_2 = 0, 0, 1, 0
PRIO_WIRELESS_CONN_DEV_3 = 0, 0, 0, 1
;
;
; SEND_CNT_WIRELESS_CONN_DEV_X
; Number of times a package should be tried to be sent over a single wireless connection.
SEND_CNT_WIRELESS_CONN_DEV_0 = 1, 0, 0, 0
SEND_CNT_WIRELESS_CONN_DEV_1 = 0, 1, 0, 0
SEND_CNT_WIRELESS_CONN_DEV_2 = 0, 0, 1, 0
SEND_CNT_WIRELESS_CONN_DEV_3 = 0, 0, 0, 1
;
;
;=============================================================================
[TransmissionConfiguration]
;
;
; RESEND_DELAY_WIRELESS_CONN_DEV_X
; Time in ms that should be waited until a package is sent again when no acknowledge is
; received per device and wireless connection.
RESEND_DELAY_WIRELESS_CONN_DEV_0 = 3, 3, 3, 3
RESEND_DELAY_WIRELESS_CONN_DEV_1 = 3, 3, 3, 3
RESEND_DELAY_WIRELESS_CONN_DEV_2 = 255, 255, 255, 255
RESEND_DELAY_WIRELESS_CONN_DEV_3 = 255, 255, 255, 255
;
;
; MAX_THROUGHPUT_WIRELESS_CONN
; Maximal throughput per wireless connection (0 to 3) in bytes/s.
MAX_THROUGHPUT_WIRELESS_CONN   = 10000, 10000, 10000, 10000
;
;
; USUAL_PACKET_SIZE_DEVICE_CONN
; Usual packet size per device in bytes if known or 0 if unknown.
USUAL_PACKET_SIZE_DEVICE_CONN = 25, 25, 25, 25
;
;
; PACKAGE_GEN_MAX_TIMEOUT
; Maximal time in ms that is waited until packet size is reached. If timeout is reached,
; the packet will be sent anyway, independent of the amount of the available data.
PACKAGE_GEN_MAX_TIMEOUT = 2, 2, 5, 5
;
;
; DELAY_DISMISS_OLD_PACK_PER_DEV
DELAY_DISMISS_OLD_PACK_PER_DEV  = 50, 50, 50, 50
;
;
; SEND_ACK_PER_WIRELESS_CONN
; To be able to configure on which wireless connections acknowledges should be sent if a
; data package has been received. Set to 0 if no acknowledge should be sent, 1 if yes.
SEND_ACK_PER_WIRELESS_CONN  = 0, 1, 0, 0
;
;
; USE_CTS_PER_WIRELESS_CONN
; To be able to configure on which wireless connections CTS for hardware flow control
; should be used. Set to 0 if it should not be used, 1 if yes.
; If enabled, data transmission is stopped CTS input is high and continued if low.
USE_CTS_PER_WIRELESS_CONN = 0, 0, 0, 0
;
;
;=============================================================================
[SoftwareConfiguration]
;
;
; TEST_HW_LOOPBACK_ONLY
; Set to 0 for normal operation, 1 in order to enable loopback on all serial interfaces
```

```
; in order to test the hardware.
TEST_HW_LOOPBACK_ONLY = 0
;
; GENERATE_DEBUG_OUTPUT
; Set to 0 for normal operation, 1 in order to print out debug infos
; (might be less performant).
GENERATE_DEBUG_OUTPUT = 1;
;
; SPI_HANDLER_TASK_INTERVAL
; Interval in [ms] of corresponding task which he will be called. 0 would be no delay -
; so to run as fast as possible.
SPI_HANDLER_TASK_INTERVAL = 5;
;
; PACKAGE_GENERATOR_TASK_INTERVAL
; Interval in [ms] of corresponding task which he will be called. 0 would be no delay -
; so to run as fast as possible.
PACKAGE_GENERATOR_TASK_INTERVAL = 5;
;
; NETWORK_HANDLER_TASK_INTERVAL
; Interval in [ms] of corresponding task which he will be called. 0 would be no delay -
; so to run as fast as possible.
NETWORK_HANDLER_TASK_INTERVAL = 5;
;
; TOGGLE_GREEN_LED_INTERVAL
; Interval in [ms] in which the LED will be turned off or on -> frequency = 2x interval
TOGGLE_GREEN_LED_INTERVAL = 500
;
; THROUGHPUT_PRINTOUT_TASK_INTERVAL
; Interval in [s] in which the throughput information will be printed out
THROUGHPUT_PRINTOUT_TASK_INTERVAL = 5
;
; SHELL_TASK_INTERVAL
; Interval in [ms] in which the shell task is called to refresh the shell
; (which prints debug information and reads user inputs)
SHELL_TASK_INTERVAL   = 10
```

Abbrevations

ALOHA	System for coordinating access to a shared communication channel
COM port	Communication Port, virtual and physical serial port interface on computer
CTS	Clear To Send, RS232 signal
ISO/OSI	7 Layers Model
RS232	Serial interface with +-12V
RTS	Ready To Send, RS232 signal
SPI	Serial Peripheral Interface, synchronous communication standard
UART	Universal Asynchronous Receiver Transmitter
UAV	Unmanned Aeral Vehicle