Lucerne University of
Applied Sciences and Arts

HOCHSCHULE
LUZERN

Technik & Architektur
FH Zentralschweiz

MSE | MASTER OF SCIENCE
IN ENGINEERING

UAV Serial Switch Documentation

Stefanie Schmidiger

MASTER OF SCIENCE IN ENGINEERING
Vertiefungsmodul I

Advisor: Prof. Erich Styger

Experte: Dr. Christian Vetterli

Horw, 2018

Hiermit erkläre ich, dass ich die vorliegende Arbeit selbstständig angefertigt und keine anderen als die angegebenen Hilfsmittel verwendet habe. Sämtliche verwendeten Textausschnitte, Zitate oder Inhalte anderer Verfasser wurden ausdrücklich als solche gekennzeichnet.

Horw, 10.01.2018 Stefanie Schmidiger

Versionen

Version 0	Initial Document	10.01.18	Stefanie Schmidiger

Abstract

With unmanned vehicles, there are always on-board and off-board components. Data transmission between those components is of vital importance. Depending on the distance between vehicle and ground station, different data transmission technologies are ideal. So far, each device was connected to a single modem and the data transmission technology used could not be switched during operation. In a previous project, a serial switch had been designed with four RS-232 interfaces that act as data input and output for devices and four RS-232 interfaces for transmitters and modems. This hardware is very flexible: data routing and transmission behavior is configurable by the user. The application running on the serial switch collects data from connected devices, puts it into a data package and sends it out via the configured transmitter. The corresponding second serial switch receives this package, extracts and verifies the payload, sends it out to the corresponding device and optionally sends an acknowledge back to the package sender.
A Teensy 3.2 development board has been used as a micro controller unit. The software was written in the Arduino IDE with the provided Arduino libraries. As the project requirements became more complex, the limit of only a serial interface available as a debugging tool became more challenging. In the end, the software ran with more than ten tasks and an overhaul of the complex structure was necessary.
This document describes the refactoring process of the previous project. In the scope of this work, an adapter board has been designed so the previous hardware could be used with the more powerful Teensy 3.5 development board and a hardware debugging interface. A new software concept for the Teensy 3.5 was developed and implemented.
The Teensy 3.5 is configured to run with FreeRTOS. The developed software uses the task scheduler and queues of the operating system to provide the same functionalities as the previous software for Teensy 3.2.
The new software concept for the Teensy 3.5 is easy to understand, maintainable and expandable. Even though the functionality of the finished project remains the same as in the first version with Teensy 3.2 and Arduino, a refactoring has been necessary. Now further improvements and extra functionalities can be implemented more easily as suggestions are given and issues are reported within this document.

Horw, January 2017 Stefanie Schmidiger

Summary

With unmanned vehicles, there are always on-board and off-board components. Data transmission between those components is of vital importance. Depending on the distance between vehicle and ground station, different data transmission technologies have to be used.
In a previous project, the hardware for a Serial Switch has been designed that features four RS-232 interfaces to connect data processing and generating devices and four RS-232 interfaces to connect modems for data transmission. The application running on the designed base board assembled data packages with the received data from its devices and sent those data packages out to the modems for transmission. The corresponding second Serial Switch received those data packages, checked them for validity and extracted the payload to send it out to its devices.
A Teensy 3.2 development board acted as the main micro controller. Teensies are small, inexpensive and powerful USB development boards for Arduino applications. The software developed was flexible and in its header files the user could configure individual baud rates for each RS-232 interface, data routing and the use of acknowledges for data packages for each modem side. The application was running with many tasks, was complex and not easy to debug because of no hardware debugging interface.
Then this follow up project was initiated with the aim of an application with better maintainability and expandability. The main requirement for this follow up project were the use of a more powerful micro controller with Free FROS as an operating system, the use of an SD card for a configuration file and data logging and a hardware debugging interface.
In the scope of this project, the Teensy 3.2 was replaced with a Teensy 3.5 development board, which featured an on-board SD card slot. The Teensy 3.5 was prepared for hardware debugging and an adapter board to use the new Teensy 3.5 with headers meant for the Teensy 3.2 was designed. This adapter board allowed the use of the same base board as was designed in the previous project.
For the Teensy 3.5 application, the concept with data packages is applied as well and the same configuration parameters are used. The configuration is read from an .ini file saved on the SD card.
The functionality of the application remains the same as in the Teensy 3.2 software but with better maintainability and an easier software concept with less tasks. Hardware debugging is now possible which is of vital importance for this application to be further expandable.
The Teensy 3.2 application was neither well documented nor running stable. While the Teensy 3.5 application provides the same functionalities as the previous software, all its issues are documented and possible workarounds are suggested. Data handling and data loss in case of unreliable data transmission channel is handled better and the application is running stable.

Table of Contents

1 Introduction

This work is being done for Aeroscout GmbH, a company that specialized in development of drones. With unmanned vehicles, there are always on-board and off-board components. Data transmission between those components is of vital importance. Depending on the distance between on-board and off-board components, different data transmission technologies have to be used.
So far, each device that generates or processes data was directly connected to a modem. This is fine while the distances between on-board and off-board components does not vary significantly. But as soon as reliable data transmission is required both in near field and far field, the opportunity of switching between different transmission technologies is vital. When data transmission with one modem becomes unreliable, an other transmission technology should be used to uphold exchange of essential information such as exact location of the drone.
The goal of this project is to provide a flexible hardware that acts as a switch between devices and modems. Data routing between all connected devices and modems should be configurable and data priority should be taken into account when transmission becomes unreliable.
It should be possible to transmit the same data over multiple modems to reduce the chance of data loss for vital information. At receiving side, this case should be handled so the original information can be reassembled correctly with the duplicated data received. In case of data loss or corrupted data, a resend attempt should be started.
The configuration should be read from a file on an SD card. This SD card should also be used to store logging data. The system should run with Free RTOS and have a command/shell interface. When no devices are connected, the Free RTOS should go into low power mode.
Data loss should be handled and encryption and interleaving should be implemented for data transmission.
A hardware should be designed that is ready for field, with a good choice of connectors, small and light weight.

It was not necessary to start from scratch for this project. Andreas Albisser has already developed a hardware with four RS-232 interfaces to connect different data generating and processing devices and four RS-232 interfaces to connect modems. As a micro controller he used the Teensy 3.2, a small, inexpensive and yet powerful USB development board that can be used with the Arduino IDE.
Andreas Albisser also developed a software for the designed UAV Serial Switch base board. The software concept implemented became more complex as the requirements were expanded during development. The finished product did not fulfill all requirements of Aeroscout GmbH. Therefore this follow up project was initiated with new requirements and the hope of a better and easier expandable software as an outcome.
Not all requirements can possibly be implemented within one semester, but good ground work should be provided for further modifications and expansions.
Because encryption requires a more powerful micro controller than has been used by Andreas Albisser, some hardware modifications are required in the scope of this project. The most profund change is the micro controller and usage of Free RTOS. This will therefore be the main focus inside the project. The aim is to have a stable application with at least as many features and working configuration parameters as the old software had.
Some requirements demand hardware changes on the base board so an evaluation needs to be done inside this project to decide how to proceed and where to invest time.
A detailed task description can be found in chapter two. An overview and critical analysis of the

hardware and software provided by Andreas Albisser is in chapter three. In chapter four, all hardware changes that have been done in the scope of this project are described, followed by chapter four with a description of the software developed. Chapter six is for the conclusion and lessons learned.

2 Task Description

This project has been done for the company Aeroscout GmbH. Aeroscout specialized in the development of drones for various needs.
With unmanned aerial vehicles, the communication between on-board and off-board devices is essential and a reliable connection for data transmission is necessary. While the drone is within sight of the control device, data can be transmitted over a wireless connection. With increasing distances, other means of transmission have to be selected such as GPRS or even satellite.
So far, the switching between different transmission technologies could not be handled automatically. The data stream was directly connected to a modem and transmitted to the corresponding receiver with no way to switch to an other transmission technology in case of data transmission failure. A visualization of this set up can be seen in Figure 2.1

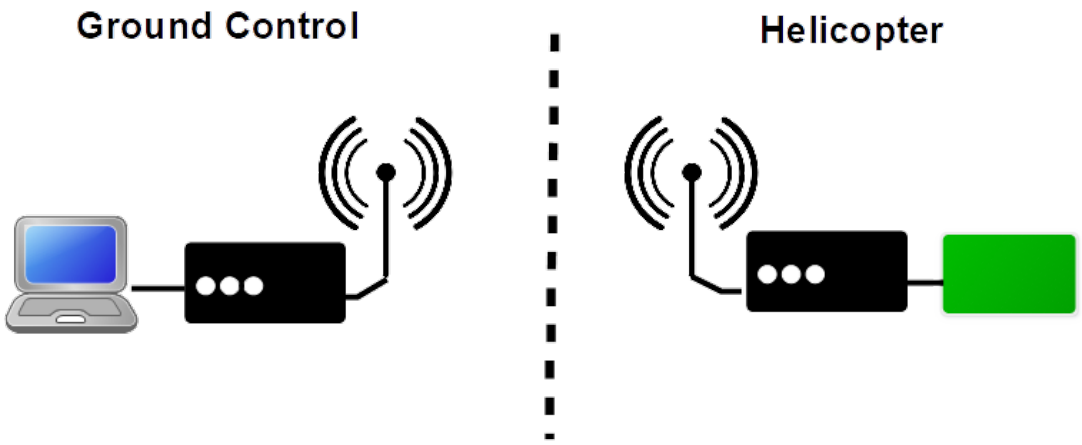


Figure 2.1: Previous system setup for data transmission

The aim of this project is to provide a solution that provides the needed flexibility. The finished product should act as a serial switch with multiple input/output interfaces for connecting devices and sensors and multiple interfaces for connecting transmitters. When one transmission technology fails to successfully transmit data, the application should automatically switch to an other transmission technology for the next send attempt. Also, multiple sensors or input streams should be able to send out data over the same wireless connection. A visualization of this set up can be seen in Figure 2.2

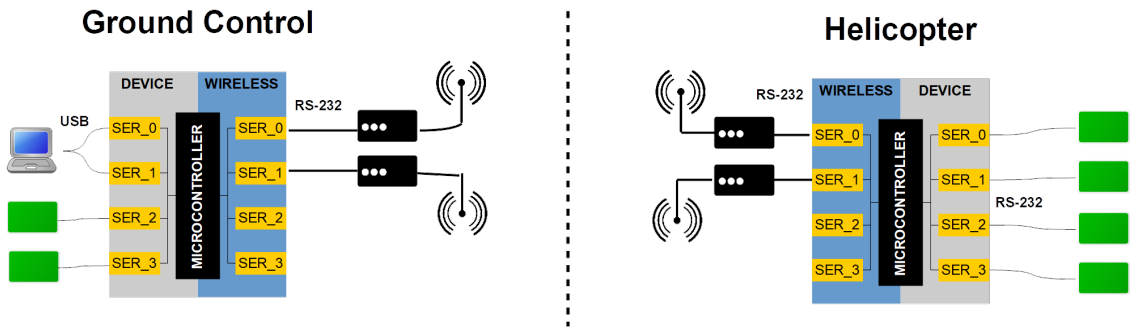


Figure 2.2: New system setup for data transmission

There are various types of information exchanged between drone and control device such as state of charge of the battery, exact location of the drone, control commandos etc. Some information such as the exact location of the drone should be prioritized over battery status information when data transmission becomes unreliable. The finished product should therefore take data priority into account.
Encryption should be configurable individually for each modem interface in case sensitive data is exchanged over a connection.
The finished product should have a hardware debugging interface such as SWD and a shell/command line interface. During run time, the software should log system data and any other relevant information to a file saved on an SD card. The SD card should also contain a configuration file so the behavior of the hardware can be changed easily.
Also, the Free RTOS should go into low power mode during idle state and software performance and memory leaks should be detected with Free RTOS Trace.
Optionally, the application should encrypt sensitive data and use the concept of interleaving to possibly recover data on receiving side in case of package loss during unreliable transmission.
A detailed description of all the requirements can be taken from appendix C.

3 Modems

3.1 RF686x RFD900x

The RF686x is a long distance radio modem to be integrated into custom projects.
Its features include:

- 3.3V UART interface
- The RTS and CTS pins are available to the user
- 5V power supply, also via USB
- Default serial port settings: 57600 baud, no parity, 8 data bits, 1 stop bit
- MAVLink radio status reporting available (RSSI, remote RSSI, local noise, remote noise)
- The RFD900x has two antenna ports and firmware which supports diversity operation of antennas. During the receive sequence the modem will check both antennas and select the antenna with the best receive signal.
- There are three different communication architectures and node topologies selectable: Peer-to-peer, multipoint network and asynchronous non-hopping mesh.
- The RFD900x Radio Modem is compatible with many configuration methods like the AT Commands and APM Planner.
- Golay error correcting code can be enabled (doubles the over-the-air data usage)
- MAVLink framing and reporting can be turned on and off.
- Encryption level either off or 128bit

The 128bit AES data encryption may be set by AT command. The encryption key can be any 32 character hexadecimal string and less and must be set to the same value on both receiving and sending modems.

3.1.1 Peer to peer network

Abb: P2P
Peer to peer network is a straight foreward connection between any two nodes. Whenever two nodes have compatible parameters and are within range, communication will succeed after they synchronize. If your setup requires more than one pair of radios within the same physical pace, you are required to set different network ID's to each pair.

3.1.2 Asynchronous non-hopping mesh

It is a straight foreward connection between two and more nodes. As long as all the nodes are within range and have compatible parameters, communication between them will succeed.

3.1.3 Multipoint network

Abb: P2MP, PTMP or PMP
In a multipoint connection, the link is between a sender and multiple receivers.

3.1.4 MAVLink

MAVLink or Micro Air Vehicle Link is a protocol for communicating with small unmanned vehicle. It is designed as a header-only message marshaling library. It is used mostly for communication between a Ground Control Station (GCS) and Unmanned vehicles, and in the inter-communication of the subsystem of the vehicle. It can be used to transmit the orientation of the vehicle, its GPS location and speed.

3.1.5 SiK

A SiK Telemetry Radio is a small, light and inexpensive open source radio platform that typically allows ranges of better than 300m “out of the box” (the range can be extended to several kilometres with the use of a patch antenna on the ground). The radio uses open source firmware which has been specially designed to work well with MAVLink packets and to be integrated with the Mission Planner, Copter, Rover and Plane.
SiK radio is a collection of firmware and tools for telemetry radios.
Hardware for the SiK radio can be obtained from various manufacturers/stores in variants that support different range and form factors. Typically you will need a pair of devices - one for the vehicle and one for the ground station.
A SiK Telemetry Radio is one of the easiest ways to setup a telemetry connection between your APM/Pixhawk and a ground station.
You can use the MAVLink support in the SiK Radios to monitor the link quality while flying, if your ground station supports it

4 Conclusion

The aim of this project was to come up with a flexible application that routes data of connected devices to modems.
There was no need to start from scratch for this project as some ground work has been done by Andreas Albisser. He developed a hardware with four RS-232 interfaces to connect data generating and processing devices and four RS-232 interfaces to connect modems for data transmission. The base board has a header to plug in a Teensy 3.2. The Teensy is a small and powerful USB development board that works with Arduino libraries and acts a main micro controller for the base board.
The software Andreas Albisser developed for the Teensy 3.2 is complex and not well thought out. Because requirements were added during development, it is hard to maintain and expand.
The task description of this project features the use of a more powerful micro controller with Free RTOS as an operating system. Using a different micro controller requires hardware changes. There were two options on how to proceed: either redesign the base board for the new micro controller or design an adapter board that routes the used signals from the new micro controller down to the header of the Teensy 3.2. Because of time reasons, the second option was chosen and an adapter board was designed for the new micro controller used.
The Teensy 3.5 was chosen to replace the Teensy 3.2. The Teensy 3.5 features an SD card slot which is also part of the requirements, has more memory and is generally more powerful. Its identical pins are backwards compatible to the pinout of the Teensy 3.2, except for the extra pins because it is slightly longer and has more pins available to the user.
The Teensy 3.5 does have hardware debugging pins available on its backside but in order to use them, the on-board bootloader has to be removed as it is in control of the debug pins and cannot be silenced.
The hardware debugging pins are routed out on a SWD debug header but the footprint of the header was faulty in the first adapter board version ordered. This mistake was corrected and a second version ordered but the second version had poor inter layer connections. The faulty footprint was tested for correctness and seemed A third version was ordered which is probably better from what can be seen under the microscope. For time reasons, it could not be assembled and tested.
Software development was started after the first Teensy Adapter Board had been assembled and the hardware debugging pinout had been corrected manually. First, the software written by Andreas Albisser was analyzed. Because there was almost no documentation available about the software concept or test results, his software concept had to reverse engineered. Because it is complex and hard to expand, a new software concept was drawn up and implemented. The new software concept is according to the ISO/OSI layers and features three main tasks: one for the physical layer (ISO/OSI layer 1), one for the data link layer (ISO/OSI layer 2) and one for the network layer (ISO/OSI layer 3).
Layer 1 deals with bytes only and communicates with the hardware components directly. Layer 2 does the assembling of data packages and splits generated data packages into bytes. Layer 3 deals with packages only, their resending in case no acknowledge was received, generates acknowledges for received packages and extracts the payload to push out to the devices connected.
Inter task communication is realized with queues, where received data and data to transmit is passed up or down the ISO/OSI layers. All task take the state of all their queues into account during runtime so they will never generate data packages when the queue they should push it to is full. This way, data is never lost intentionally. Currently, each task may lose data unintentionally if a queue operation fails without it being full or empty but for apparently no reason. The only task that will drop data on purpose is the physical layer as it will flush a certain amount of bytes from its queue when full if new data is received and old data has not yet been processed.

Generally, the implementation provides about the same functionality as the software developed by Andreas Albisser. The only parameter missing that was part of Andreas' configuration is the limitation of throughput per wireless connection.
The base provided by this software is well documented and suitable for further expansion which was not the case with the previous software written for the Teensy 3.2.
The next step would be to implement package numbering instead of a system time stamp in the package header. Currently, the receiver does not know about missing packages when looking at the time stamp in the package header because the time stamp is not monotonically increasing. To ensure that packages are always received in the right order, either the sender waits for the acknowledge for a package before sending out the next one or the receiver implements a queue to reassemble packages in their right order before extracting the payload and sending it out to the correct device.
Aeroscout GmbH would also like to have data priority configurable. Currently, there is no configuration parameter that allows the user to prioritize one connected device over an other in case of unreliable wireless connection between on-board and off-board Serial Switch.
Also, logging has not been implemented yet because of time reasons. The Serial Switch currently prints out debug information on the shell but does not save it in a file.
Data encryption and interleaving were part of the task description but have not been implemented yet either.
Generally, the aim was of this project was to provide a software that works at least as good as the one provided by Andreas Albisser. Documentation any testing was done at the very end because development of a good and reliable product was prioritized.
When testing Andreas Albisser's application with an autopilot software as in a real use-case of Aeroscout GmbH, a connection could not be established successfully when acknowledges were configured for the application and on-board and off-board Serial Switches were communicating wirelessly.
When testing the new application with the same autopilot software, a connection was established successfully, even with acknowledges configured and wireless communication between the off-board and on-board devices.
Therefore, the goal of this project has been reached because the outcome seems to be of better quality than the state of the project upon start.
More time should have been invested into the testing phase to get more detailed results of the project outcome.

4.1 Lessons Learned

This project has been educational in many ways. It was my first time using the operating system Free RTOS and I was struggling during the starting phase because of too many new tools.
Also because the start of the project was rather slow and it felt like there was no real progress to show, I was neglecting the documentation up until the very end of the semester. This results in missing pictures in the documentation because they were not taken at that time and missing information because not all the configurations could be remembered when documenting a test case.
During the semester, I spent most time working on the project, implementing features and improving performance. This left me with little time for testing at the end where I should have stopped development earlier to invest more time in the testing phase so the person to follow up with this project would know exactly where he/she is at.
I do not have a lot of experience with schematics and layouts and did not know about the very high possibility of poor inter layer connections with internally ordered PCBs. Altium Designer together with finding the mistake in the poor inter layer connections cost me a lot of time during this project.
Also, I am a first time LaTex user and there were more than one occasion where I remembered Erich

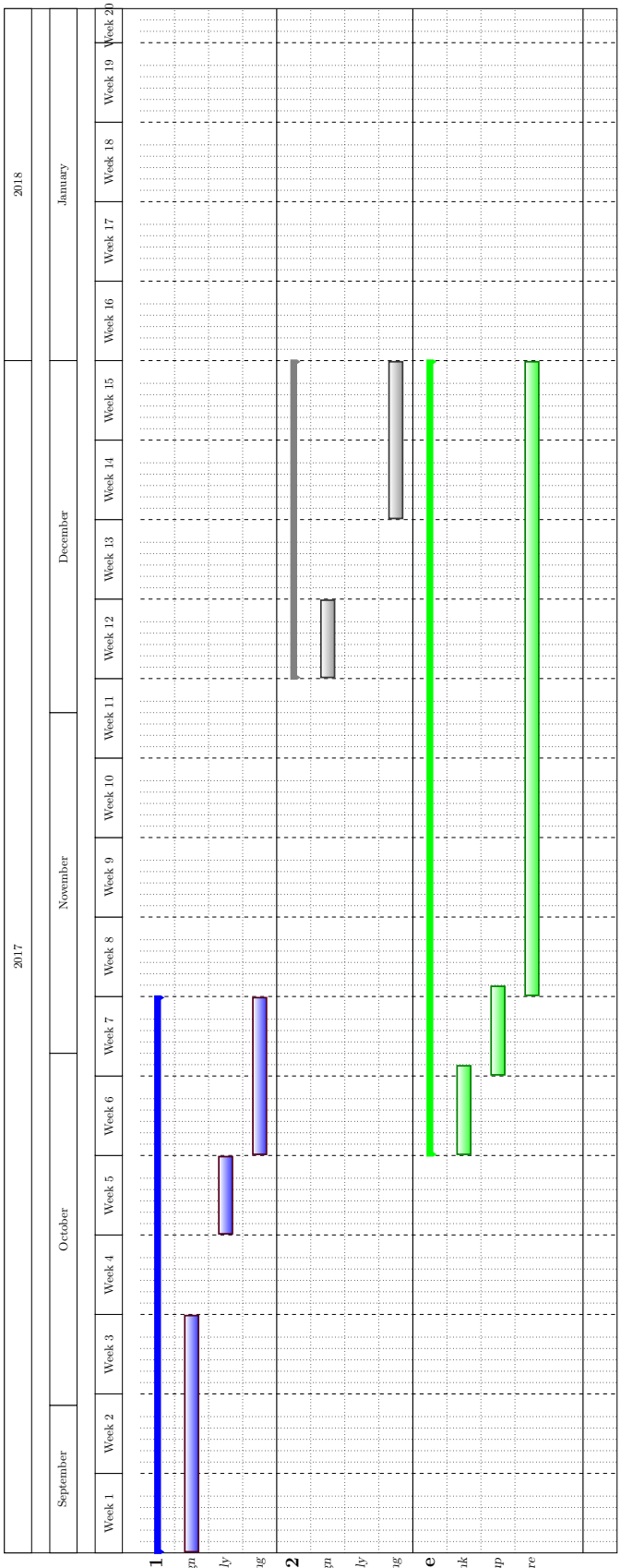
Styger’s advice: "LaTex without version control is suicide". I can only agree with this statement and am forever glad he mentioned it so early.
Last but not least, it will be my goal to always keep the requirements in mind at all times during the next project. It is easy to get lost in coding and adding features. One should always take a step back, look at the requirements and the task description to make sure no element is forgotten. And maybe have a look at the grading paper to see where most time should be invested.

References

Abbrevations

ALOHA	System for coordinating access to a shared communication channel
COM port	Simulated serial interface on computer
GND	Ground reference, usually 0V
HSLU	Lucerne University of Applied Sciences and Arts
HW	Hardware
ISO/OSI	7 Layers Model
MCU	Micro Controller Unit
PC	Personal Computer
PCB	Printed Circuit Board
RS-232	Serial interface with +-12V
RTT	Real Time Transfer, Segger terminal
RTOS	Realtime Operating System
RX	Received signal
SPI	Serial Peripheral Interface, synchronous communication standard
SW	Software
SWD	Serial Wire Debug, hardware debugging interface
TX	Transmitted signal
TTL	Transistor Transistor Level, 5V level
UART	Universal Asynchronous Receiver Transmitter
UAV	Unmanned Aeral Vehicle
USB	Universal Serial Bus

Anhang A Project Plan for Complete Hardware Redesign



Anhang B Configuration File

A sample configuration file saved on the SD card looks as 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]
20 ;
21 ;
22 ; PRIO_WIRELESS_CONN_DEV_X
23 ; Priority of the different wireless connections from the viewpoint of a single device.
24 ; 0: Wireless connection is not used; 1: Highes priority; 2: Second priority, ..
25 PRIO_WIRELESS_CONN_DEV_0 = 1, 0, 0, 0
26 PRIO_WIRELESS_CONN_DEV_1 = 0, 1, 0, 0
27 PRIO_WIRELESS_CONN_DEV_2 = 0, 0, 1, 0
28 PRIO_WIRELESS_CONN_DEV_3 = 0, 0, 0, 1
29 ;
30 ;
31 ; SEND_CNT_WIRELESS_CONN_DEV_X
32 ; Number of times a package should be tried to be sent over a single wireless connection.
33 SEND_CNT_WIRELESS_CONN_DEV_0 = 1, 0, 0, 0
34 SEND_CNT_WIRELESS_CONN_DEV_1 = 0, 1, 0, 0
35 SEND_CNT_WIRELESS_CONN_DEV_2 = 0, 0, 1, 0
36 SEND_CNT_WIRELESS_CONN_DEV_3 = 0, 0, 0, 1
37 ;
38 ;
39 ;=============================================================================================
40 [TransmissionConfiguration]
41 ;
42 ;
43 ; RESEND_DELAY_WIRELESS_CONN_DEV_X
44 ; Time in ms that should be waited until a package is sent again when no acknowledge is
45 ; received per device and wireless connection.
46 RESEND_DELAY_WIRELESS_CONN_DEV_0 = 3, 3, 3, 3
47 RESEND_DELAY_WIRELESS_CONN_DEV_1 = 3, 3, 3, 3
48 RESEND_DELAY_WIRELESS_CONN_DEV_2 = 255, 255, 255, 255
49 RESEND_DELAY_WIRELESS_CONN_DEV_3 = 255, 255, 255, 255
50 ;
51 ;
52 ; MAX_THROUGHPUT_WIRELESS_CONN
53 ; Maximal throughput per wireless connection (0 to 3) in bytes/s.
54 MAX_THROUGHPUT_WIRELESS_CONN   = 10000, 10000, 10000, 10000
55 ;
56 ;
57 ; USUAL_PACKET_SIZE_DEVICE_CONN
58 ; Usual packet size per device in bytes if known or 0 if unknown.
59 USUAL_PACKET_SIZE_DEVICE_CONN = 25, 25, 1, 1
60 ;
```

```
;
; 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, 20, 20
;
;
; DELAY_DISMISS_OLD_PACK_PER_DEV
DELAY_DISMISS_OLD_PACK_PER_DEV  = 10000, 10000, 10000, 10000
;
;
; 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
```

Anhang C Task Description

The full task description for this project can be found below.

MSE – Vertiefungsmodul 1

Horw, 17.Sept.2017

Aufgabenstellung für:

Stefanie Schmidiger_____ (Masterstudierende/r)

Embedded Systems und Mikroelektronik_____ (Fachgebiet)

von Prof. Erich Styger_____ (Advisor)

Dr. Christian Vetterli_____ (Experte/Expertin)

1. Arbeitstitel

UAV Serial Switch

2. Fremdmittelfinanziertes Forschungs-/Entwicklungsprojekt

KTI Projekt LINDA „UAV Power Line Inspektion“

3. Industrie-/Wirtschaftspartner

Aeroscout GmbH, ewz

4. Fachliche Schwerpunkte

Schwerpunkt A: Mikrokontroller
Schwerpunkt B: Kommunikation und Bus-Schnittstellen
Schwerpunkt C: Sensoren und Sensorik

5. Einleitung

In autonomen Flugsystemen besteht der Bedarf an einer universellen und sicheren seriellen Verbindung. Diese wird sowohl für interne Board-Systeme als auch zur Kommunikation nach aussen benötigt. Es existiert eine Vorarbeit von Andreas Albisser. Diese soll in dieser Arbeit verbessert und feldtauglich gemacht werden.

6. Aufgabenstellung

Definieren und Verfeinern Sie in Zusammenarbeit mit dem Industriepartner die Anforderung. Die Basisanforderung sind die folgenden:

- Hardware
  - Optimierung Grösse und Gewicht
  - Feldtauglichkeit (Stecker/Anschlüsse/Gehäuse)

  - Leistungsfähigeren Prozessor mit mehr Speicher und RNG/Encryption Support (z.B. K64 oder K66).
  - SWD/JTAG Debugging
  - UART Hardware Flow Control
  - SD Karte (Normal oder Micro)
- Software
  - FreeRTOS als Betriebssystem
  - Low Power: Tickless IDLE Mode mit einfachem IDLE Sleep Mode
  - Shell/Command Line Interface
  - Verifikation mit FreeRTOS+Trace
  - SD-Karte/File System für Logging und Konfiguration (Schlüssel)
  - Behandlung verlorener Datenpakete: Kodierung, Überlagerung/Kodierung

Die Hard- und Software soll zuverlässig in einer Feldumgebung funktionieren. Überlegen Sie sich auch mögliche Lösungen für eine sichere Verbindung (Verschlüsselung). Erstellen Sie einen Projektplan mit den nötigen Meilensteinen. Verifizieren und Testen Sie Ihre Lösung und dokumentieren Sie sowohl mit einer Projektdokumentation (Bericht) als auch mit einem Benutzerhandbuch.

---

7. Durchführung der Arbeit

Termine

Start der Arbeit:	Montag, 17.Sept.2017
Zwischenpräsentation:	Nach Absprache mit Advisor/Experten, Nov. 2017
Schlusspräsentation:	Nach Absprache mit Advisor/Experten, Januar 2018
Abgabe Bericht:	bis Fr. 22.12.2017 – 16.30 Uhr (D311, Prof. Erich Styger)

Organisatorisches

Advisor und Masterstudierende vereinbaren ein wöchentliche Besprechung.

Die Termine für die Präsentationen (Zwischen- und Schlusspräsentation) werden frühzeitig vereinbart.

---

8. Dokumentation

Die wissenschaftliche Dokumentation ist in 3-facher Ausführung zu erstellen.

- die folgende Selbstständigkeitserklärung auf der Rückseite des Titelblattes:
  „Hiermit erkläre ich, dass ich die vorliegende Arbeit selbstständig angefertigt und keine anderen als die angegebenen Hilfsmittel verwendet habe. Sämtliche verwendeten Textausschnitte, Zitate oder Inhalte anderer Verfasser wurden ausdrücklich als solche gekennzeichnet.
  Horw, Datum, eigenhändige Unterschrift"
- Inhaltsverzeichnis.
- eine Zusammenfassung maximal 1 A4.
- einen englischen Abstract maximal 1 A4.
- Kurzlebenslauf maximal 1 A4 (tabelarisch).

Zusätzlich muss dem Advisor eine CD mit dem Bericht (inkl. Anhänge), mit den Präsentationen, Messdaten, Programmen, Auswertungen, usw. abgeben werden.

9. Fachliteratur/Web-Links/Hilfsmittel

Vorarbeiten (Software, Layout, PCB's) von Andreas Albisser.

10. Zusätzliche Bemerkungen

- keine

11. Beilagen

- Bewertungsraster
- Hinweise zu Projektarbeiten

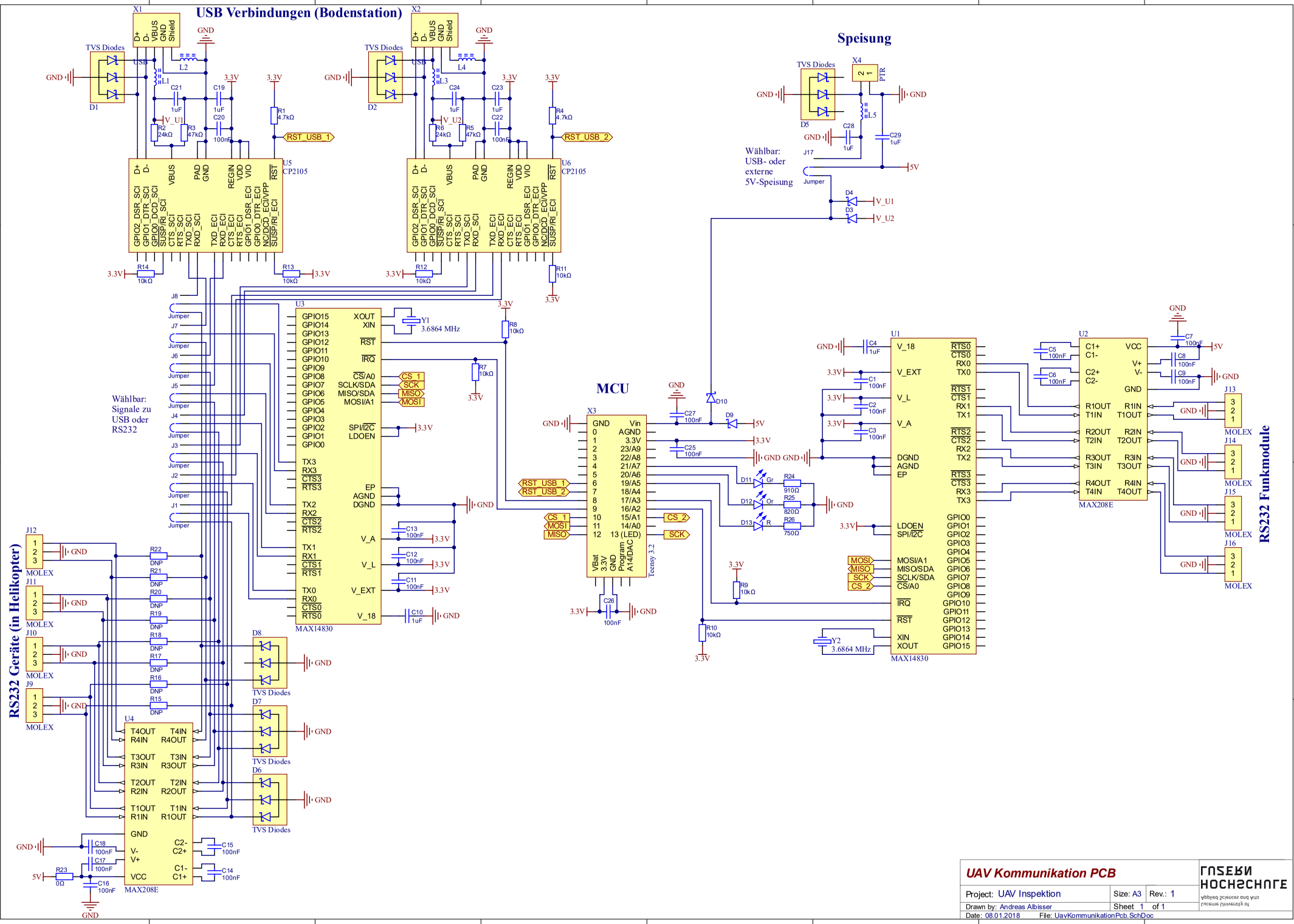
Horw, 18.Sept.2017

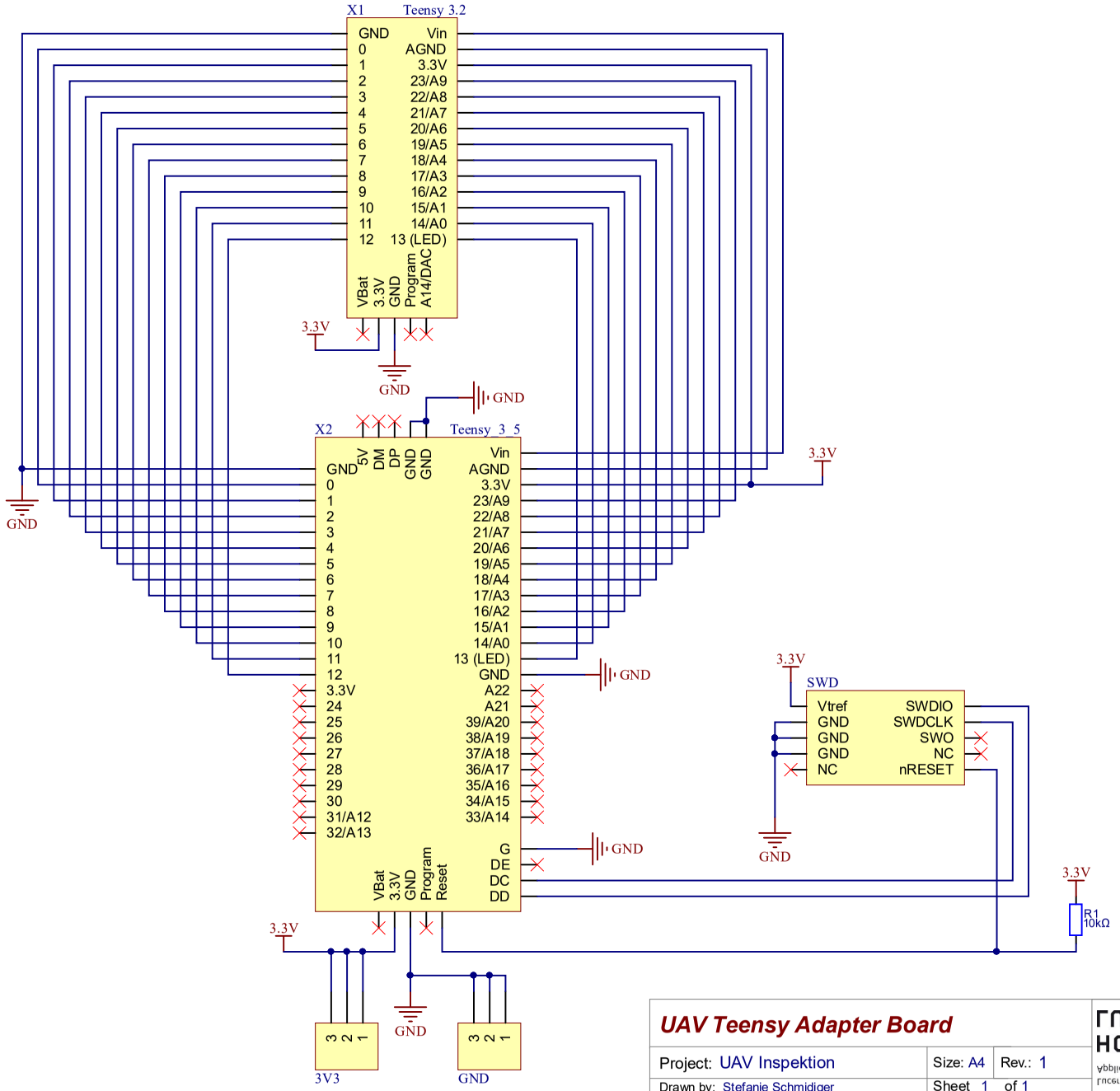
Advisor	Experte/Expertin	Studierende
_____	_____	_____

→ eine Kopie der Aufgabenstellung ist vor Semesterbeginn an den Studiengangleiter abzugeben!

Anhang D Schematics

The schematics of both the base board designed by Andreas Albisser and the Teensy adapter board can be found in this section.


USB Verbindungen (Bodenstation)
Speisung
MCU
RS232 Geräte (in Helikopter)
RS232 Funkmodule
Wählbar:
Signale zu
USB oder
RS232
Wählbar:
USB- oder
externe
5V-Speisung
TVS Diodes
CP2105
MAX14830
MAX208E
Teensy 3.2
3.6864 MHz
UAV Kommunikation PCB
Project: UAV Inspektion
Size: A3
Rev.: 1
Drawn by: Andreas Albisser
Sheet 1 of 1
Date: 08.01.2018
File: UavKommunikationPcb.SchDoc



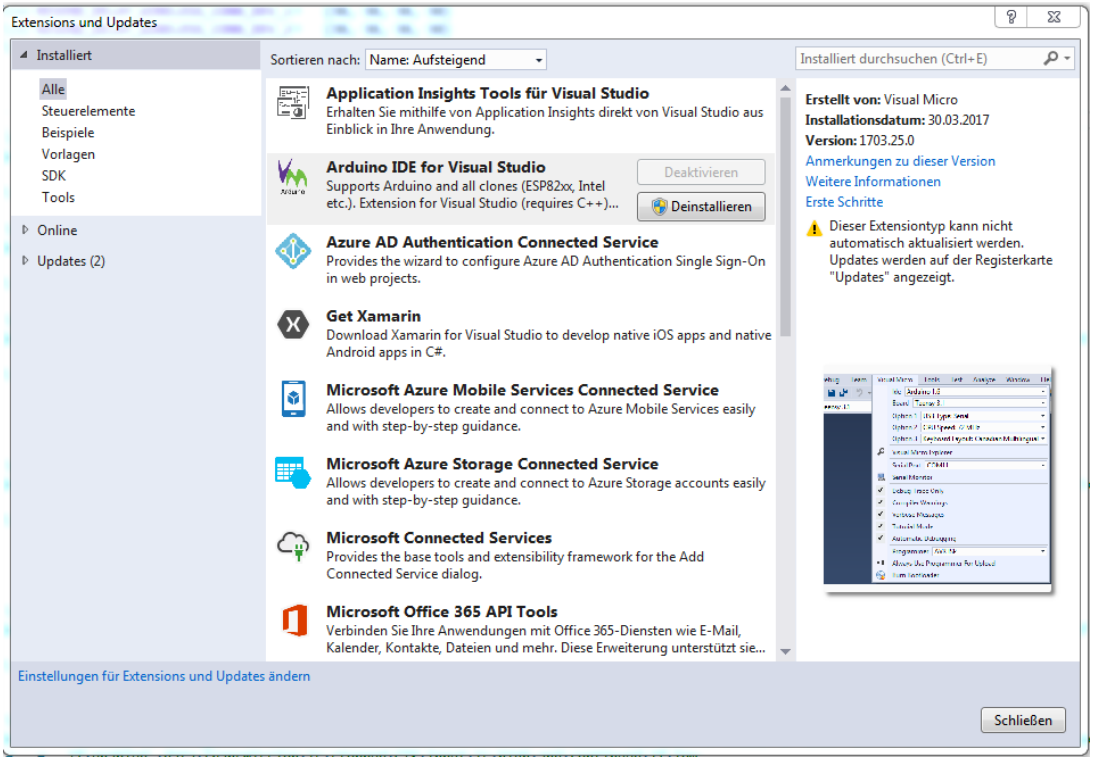
UAV Teensy Adapter Board			HOCHSCHULE
Project: UAV Inspektion		Size: A4	Rev.: 1
Drawn by: Stefanie Schmidiger		Sheet 1 of 1	
Date: 08.01.2018	File: Teensy36AdapterBoard.SchDoc		

Anhang E Provided Documentation

All documentation provided by Andreas Albisser can be found in this section. There are two documents, an installation guide for all software needed and the documentation of his work.

Aufsetzen der Softwareentwicklungsumgebung für den UAV Serial Switch

1. Installation Visual Studio Enterprise 2015 (Version 14)
2. Da einige Libraries von hier verwendet werden, muss man leider auch die Arduino DIE installieren (zwingend die alte Version 1.8.1): https://www.arduino.cc/en/main/software
3. Für den Support des Verwendeten Mikrocontroller-Boards muss Teensduino installiert werden: https://www.pjrc.com/teensy/td_download.html
4. Danach muss im Visual Studio noch die Extension von Visual Micro installiert werden: In Visual Studio via Extras => Extensions und Updates



5. Danach müssen via den Visual Micro Explorer (vMicro => Visual Micro Explorer) noch die beiden verwendeten Libraries „Queue by SMFSW“ und „TaskScheduler by Anatoli Arkhipenko“ installiert werden (Reiter „Manage Libraries“ im Visual Micro Explorer).
6. Das Projekt sollte nun erfolgreich Kompiliert und hinuntergeladen werden können.

UAV Serial Switch

1 Funktionalität

Der Serial Switch besitzt grundsätzlich zwei Seiten – die Device Seite sowie die Wireless Seite. Auf Device Seite befindet sich im Falle der Bodenstation die Laptops oder im Falle des Fluggerätes die verschiedenen Komponenten, zu denen eine serielle Verbindung besteht. Auf Wireless Seite befinden sich jeweils die Funkmodule. Sowohl auf Device- als auch auf Wireless-Seite sind jeweils vier Verbindungen vorhanden, welche SER_0 bis SER_3 benannt sind. In Abbildung 1 sieht man die Übersicht über die involvierte Hardware in einem typischen Use Case.

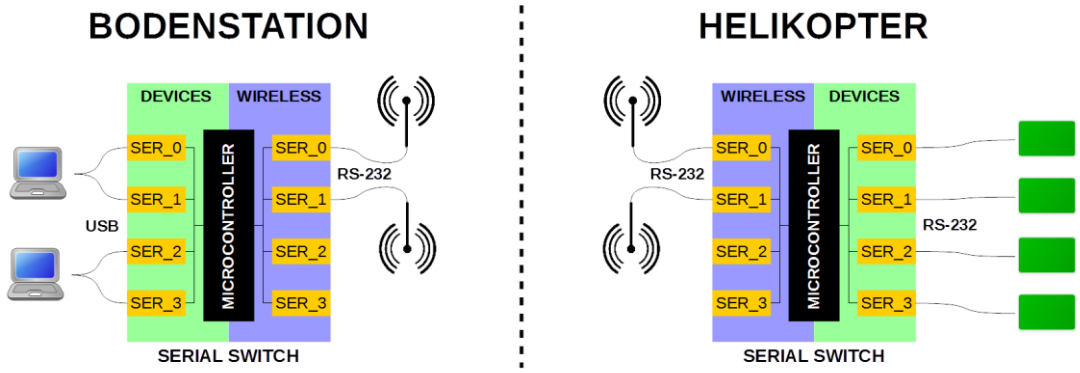


Abbildung 1: Übersicht Hardware, typischer Use Case

Sowohl bei der Bodenstation als auch im Helikopter befindet sich dieselbe Hardware. Auf Device-Seite kann auf dem Serial Switch mit Jumpern gewählt werden, ob die Daten per RS-232 oder USB CDC abgegriffen werden.

Beim Serial Switch muss jeweils unter anderem Konfiguriert werden, welche Serielle Verbindung von der Device Seite mit welcher Priorität über welchen Funkkanal übertragen werden soll. Dies ist in Kapitel 2 genauer erläutert.

2 Konfiguration und Priorisierung

Konfigurationsparameter	Beschreibung	Wertebereich/Beispiel
BAUD_RATES_WIRELESS_CONN sowie *BAUD_RATES_DEVICE_CONN*	Konfiguration der Baudraten der einzelnen seriellen Verbindungen.	{*SER_0, SER_1, SER_2, SER_3*} Es wird jeweils direkt die Baudrate eingefüllt. Momentan werden die Baudraten 9600, 38400, 57600 sowie 115200 unterstützt. *Beispiel*: {9600, 38400, 57600, 115200}
PRIO_WIRELESS_CONN_DEV_X *(X = 0-3)*	Priorität einer Wireless Verbindung aus Sicht einer Device Verbindung. Dieser Konfigurationsparameter existiert für jedes Device und gibt an, mit welcher Priorität welche Wireless Verbindung genutzt werden soll. Er besteht entsprechend aus vier Werten, wobei der Wert Null bedeutet, dass die entsprechende Verbindung nicht genutzt werden soll. Wichtig: Die Prioritäten müssen aufeinanderfolgend sein, wenn also nur die Prioritäten 1 und 3 vorhanden sind, wird die Priorität 3 nie erreicht.	{*SER_0, SER_1, SER_2, SER_3*} Wertebereich: 0-4 0: Verbindung nicht verwenden 1: Höchste Priorität 2: Zweite Priorität usw. Ist bei zwei Verbindungen dieselbe Priorität vorhanden, so werden die Daten über beide Verbindungen verschickt. *Beispiel*: {1, 2, 0, 0} Hier wird die Wireless Verbindung *SER_0* mit höchster Priorität verwendet, *SER_1* mit der zweiten Priorität. *SER_2* und *SER_3* werden nicht verwendet.
SEND_CNT_WIRELESS_CONN_DEV_X *(X = 0-3)*	Definiert die Anzahl Sendeversuche pro Wireless-Verbindung, die durchgeführt werden, bevor auf eine Verbindung mit niedriger Priorität ausgewichen wird oder aber die Daten verworfen werden, wenn keine Verbindung mit niedrigerer Priorität vorhanden ist.	{*SER_0, SER_1, SER_2, SER_3*} Wertebereich: 0-255 Definiert die Anzahl Versuche pro Wireless Verbindung, die durchgeführt werden, die Daten über diese Verbindung zu senden. *Beispiel*: {5, 10, 0, 0} Bei der Verbindung *SER_0* werden 5 Versuche durchgeführt, die Daten zu senden, bei *SER_1* 10 versuche sowie bei den Verbindungen *SER_2* und *SER_3* keine Versuche.
RESEND_DELAY_WIRELESS_CONN_DEV_X	Definiert pro Device und Wireless Connection die Zeit in ms, die gewartet wird, bis das Paket erneut gesendet wird falls in dieser Zeit keine Empfangsbestätigung empfangen wird. Hinweis: Ein zu kurzes Resend-Delay sollte vermieden werden, da ansonsten Pakete mehrfach gesendet werden bevor das Acknowledge überhaupt empfangen wird.	{*SER_0, SER_1, SER_2, SER_3*} Wertebereich: 0-255 *Beispiel*: *RESEND_DELAY_WIRELESS_CONN_DEV_0* {10, 20, 20, 20} Beim Device 0 wird bei der Wireless Verbindung 0 10ms gewartet, bis erneut ein Paket gesendet wird, bei den anderen Verbindungen jeweils 20ms.
MAX_THROUGHPUT_WIRELESS_CONN	Beschränkung des maximalen Durchsatzes pro Wireless Verbindung in bytes/s. Haben mehrere Devices bei der gleichen Wireless Verbindung dieselbe Priorität (*PRIO_WIRELESS_CONN_DEV_X*) und reicht der Durchsatz aufgrund davon nicht aus, so werden die Daten des Devices mit der niedrigeren Devicepriorität auf eine andere Verbindung umgeleitet oder aber verworfen (abhängig von den Parametern *PRIO_WIRELESS_CONN_DEV_X* und *SEND_CNT_WIRELESS_CONN_DEV_X*)	{*SER_0, SER_1, SER_2, SER_3*} Wertebereich: 0-(2^{32}-1) bytes/s *Beispiel*: {1000, 3000, 0, 0} *SER_0:* Maximal 1000 bytes/s *SER_1:* Maximal 3000 bytes/s *SER_2:* Kein Funkmodul vorhanden *SER_3:* Kein Funkmodul vorhanden
USUAL_PACKET_SIZE_DEVICE_CONN	Konfiguration der üblichen Paketgrösse auf Device Seite falls bekannt. In der Software wird dann maximal *PACKAGE_GEN_MAX_TIMEOUT* gewartet, ganze Devicepakete zu sammeln und in Wirelesspakete zu verpacken. Ist dieser Wert 0, so mit einem Intervall von 1ms die vorhandenen Daten in ein Paket verpackt und verschickt.	{*SER_0, SER_1, SER_2, SER_3*} Übliche Paketgrösse in bytes pro Deviceverbindung falls bekannt oder 0 falls unbekannt. Wertebereich: 0-512 bytes *Beispiel*: {256, 0, 0, 0} *SER_0:* Übliche Paketgrösse 256 bytes *SER_1-3:* Übliche Paketgrösse unbekannt
PACKAGE_GEN_MAX_TIMEOUT	Maximale Zeit, die gewartet wird, zwischen dem Zeitpunkt von dem Daten vom Device	{*SER_0, SER_1, SER_2, SER_3*} Maximales Timeout pro Device in ms.

	vorhanden sind bis die Daten versendet werden ohne *USUAL_PACKET_SIZE_ DEVICE_CONN* erreicht wurde.	*Beispiel*: {3, 3, 3, 3} Maximales Timeout 3ms für alle Devices.
DELAY_DISMISS_OLD_ PACK_PER_DEV	Legt fest, wie lange ein Paket im Buffer behalten wird, währen bereits ein neues Paket vom selben Device darauf wartet, gesendet zu werden.	{*SER_0, SER_1, SER_2, SER_3*} Zeit in ms, während der versucht wird, ein altes Paket erneut zu senden, während bereits ein neues Paket vom selben Device vorhanden ist. Beispiel: {5, 10, 5, 5} Beim Device *SER_1* wird 10ms gewartet, bei den anderen Devices 5ms.
SEND_ACK_PER_ WIRELESS_CONN	Parameter um zu konfigurieren, bei welchen Wireless Verbindungen bei Empfang eines Datenpaketes ein Acknowledge zurückgesendet werden soll.	{*SER_0, SER_1, SER_2, SER_3*} 1 für senden von Acknowledge, bei 0 wird kein Acknowledge gesendet. Beispiel: {1, 0, 1, 1} Bei allen Verbindungen mit Ausnahme von *SER_1* werden Acknowledges zurückgesendet.
USE_CTS_PER_WIRELESS_ CONN	Parameter zum Ein-/Ausschalten der Berücksichtigung des CTS-Signals (Hardware Flow Control) auf Wireless-Seite (ist das Signal 0, so werden Daten gesendet, wenn 1 wird gewartet => Input auf Serial Switch Hardware). Hinweis: Wenn dies eingeschalten wird muss zwingend auch die entsprechende Hardware-Verbindung zum Wireless Transmitter gemacht werden, da ansonsten gar keine Daten mehr gesendet werden (es ist ein schwacher interner Pull Up Widerstand dieses Signals vorhanden).	{*SER_0, SER_1, SER_2, SER_3*} 1 für Berücksichtigung des CTS-Signals, 0 wenn nicht. Beispiel: {1, 0, 0, 0} Bei der Verbindung *SER_0* wird das CTS-Signal berücksichtigt, bei allen anderen nicht.

Tabelle 1: Auflistung und Beschreibung der verschiedenen Konfigurationsparameter

In Tabelle 1 sind die verschiedenen Konfigurationsparameter aufgelistet und beschrieben. In der Firmware finden sich diese im File *serialSwitch_Config.h*, momentan werden diese Parameter statisch zur Compilezeit konfiguriert.

Neben der in der Tabelle beschriebene *PRIO_WIRELESS_CONN_DEV_X* existiert auch noch die Devicepriorität, welche höher gewichtet wird. Die Devicepriorität ist fix abhängig vom verwendeten seriellen Anschluss des Devices. So wird dem Anschluss *SER_0* die höchste Priorität zugeordnet, *SER_1* die zweithöchste usw. Entsteht also beispielsweise ein Engpass beim Datendurchsatz, so werden die Daten des Devices mit der höheren Devicepriorität bevorzugt.

Nicht unterschätzt werden sollte das Timing, in welchem Abstand die Sendeversuche (wenn Parameter *SEND_CNT_WIRELESS_CONN_DEV_X* > 1) vorgenommen werden, siehe Parameter *RESEND_DELAY_PER_DEVICE_MS*. Ist dieser Parameter zu klein gewählt, so werden mehrere Pakete gesendet, bevor überhaupt die Empfangsbestätigung empfangen wird, was zu unnötig hohem Traffic über die Wireless Verbindung führen kann. Wie gross dieser Parameter sein sollte hängt unter anderem von der verwendeten Funkverbindung und der Baudrate ab.

Ein anderer wichtiger Parameter bezüglich des Timings ist *PACKAGE_GEN_MAX_TIMEOUT*. Damit wird definiert, wie lange maximal bei einem Device auf Daten gewartet wird, bis ein Paket daraus generiert und danach verschickt wird. Ist dies zu kurz gewählt, so kann dies die Anzahl Pakete bei der gleichen zu übertragenden Anzahl Bytes von Device zu Device erhöhen. Dadurch wird unnötig viel Traffic auf der Wireless Verbindung generiert. Bei einer langen Zeitdauer wiederum werden die

Daten entsprechend länger zwischengespeichert, wodurch die Verzögerung auf dem Kommunikationspfad zunimmt.

3 Beschreibung Wireless Kommunikation

Eine grobe Übersicht über die Hardware wurde bereits in Kapitel 1 gegeben. Die Frage ist nun natürlich, wie die in Kapitel 2 beschriebenen Funktionalitäten in der Software umgesetzt werden. Grundsätzlich geht es darum, dass die Daten unverfälscht vom Device der Bodenstation unverfälscht zum Device des Helikopters übertragen werden zu können und umgekehrt. Dazu erhalten die Daten der Devices für die Wireless Strecke einen Header gemäss Tabelle 2. Der Erhalt der Daten wird bestätigt, indem die Gegenseite eine Empfangsbestätigung zurücksendet.

Name Feld	**Beschreibung**	**Länge [bytes]**
PACK_START	Escape-Character (\e in C, 0x1B ASCII). Bezeichnet den Beginn eines Paketes.	1
PACK_TYPE	Markierung des Pakettyps: - 0x01: Datenpaket - 0x02: Empfangsbestätigung für Paket	1
DEV_NUM	Nummer des Devices, welchem dieses Paket zugeordnet ist (0 bis 3)	1
SESSION_NR	Session Nummer. Werden Pakete mit einem älterem Zeitstempel empfangen, als vorher bereits empfangen wurden, werden diese verworfen – ausser wenn die Session Nummer gewechselt hat. Die Session Nummer wird jeweils beim Boot Up zugewiesen (Zufallszahl).	1
SYS_TIME	Systemzeit in ms. Wird auch zur eindeutigen Identifikation eines Paketes verwendet.	4
PAYLOAD_SIZE	Grösse des Payloades in bytes.	2
CRC8_HEADER	8 bit Checksumme über den Header zur Verifikation der Gültigkeit des Inhaltes.	1
PAYLOAD	Der Inhalt des Payloads ist abhängig vom Pakettyp: - *PACK_ TYPE* 0x01 (Datenpaket): In diesem Fall beinhaltet der Payload die Daten, welche zwischen dem Device der Bodenstation sowie dem Device des Helikopters ausgetauscht werden sollen. - *PACK_TYPE* 0x02 (Empfangsbestätigung Paket): Beinhaltet genau eine *SYS_TIME* des Paketes, dessen Empfang bestätigt werden soll.	n
CRC16_PAYLOAD	16 bit Checksumme über den Payload.	2
Bei allen Feldern mit Ausnahme von *PACK_START* wird, falls der Escape-Character vorkommt (\e), dieser durch "\e" ersetzt (ASCII: 0x22 0x1B 0x22). Dies wird auf Empfangsseite entsprechend wieder rückgängig gemacht. Somit wird verhindert, dass sich in den Daten per Zufall ein gültiger Header zusammensetzt. (Ein allfälliges "\e" in den Daten wird entsprechend durch ein ""\e"" ersetzt usw.)		

Tabelle 2: Beschreibung eines Paketes zur Kommunikation auf Wireless Seite

Der Fluss der Daten innerhalb eines Serial Switches ist in Abbildung 2 aufgezeichnet. Die Funktion der verschiedenen Softwarekomponenten wird nachfolgend kurz erläutert:

- **HW Buffer Interface**: Liest und schreibt die Daten von und zu den Hardware Buffern unter Berücksichtigung des Parameters *MAX_THROUGHPUT_WIRELESS_CONN* auf Wireless Seite. Die Daten werden von den entsprechenden Queues gelesen respektive geschrieben.
- **Package Generator**: Generiert einerseits Datenpakete gemäss Tabelle 2 aus den Daten, die von den Devices gelesen wurden unter Berücksichtigung der Parameter

USUAL_PACKET_SIZE_DEVICE_CONN sowie *PACKAGE_GEN_MAX_TIMEOUT*. Andererseits werden Pakete zur Empfangsbestätigung generiert.

- **Wireless Ack Send Handler**: Liest die generierten Acknowledge Pakete und schreibt diese zum HW Buffer Interface. Empfangsbestätigungspakete werden einmalig gesendet.
- **Wireless Data Send Handler**: Liest die generierten Datenpakete ein, diese werden unter Berücksichtigung der Parameter *PRIO_WIRELESS_CONN_DEV_X* an die konfigurierte Wireless Verbindung oder Verbindungen gesendet. Danach wird das Paket noch im Speicher behalten, bis die Empfangsbestätigung eintrifft. Falls diese nicht eintrifft, wird das Paket entsprechend dem Parameter *SEND_CNT_WIRELESS_CONN_DEV_X* nochmals gesendet oder aber verworfen.
- **Wireless Package Extractor**: Liest die empfangenen Wirelessdaten ein und überprüft anhand der Checksummen, ob die Daten fehlerfrei sind. Sind sie fehlerfrei, so werden – je nach Pakettyp, siehe Tabelle 2 – die Daten via dem HW Buffer Interface an die Devices weitergeleitet. Bei erfolgreich empfangenen Datenpaketen wird dem Package Generator mitgeteilt, ein Empfangsbestätigungspaket zu generieren. Erfolgreich empfangene Empfangsbestätigungen werden dem Wireless Send Handler weitergeleitet.

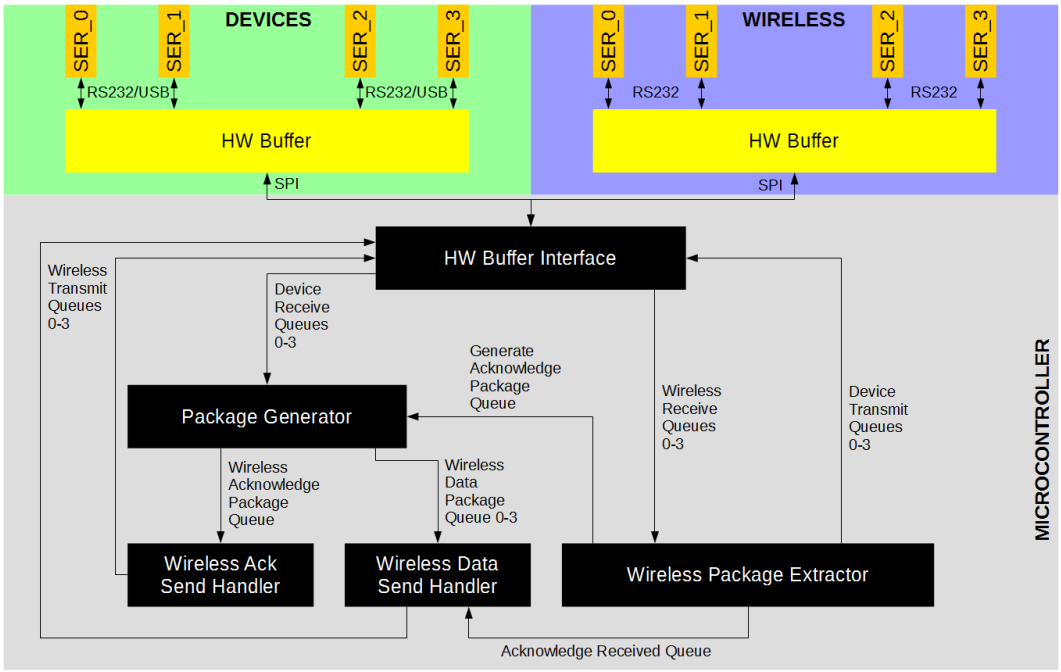


Abbildung 2: Detaillierte Systemübersicht eines einzelnen Serial Switches inklusive Softwarekomponenten

Lebenslauf

Personalien

Name	Stefanie Schmidiger
Adresse	Gutenegg 6125 Menzberg
Geburtsdatum	30.06.1991
Heimatort	6122 Menznau
Zivilstand	ledig

Ausbildung

August 1996 - Juli 2003	Primarschule, Menzberg
August 2003 - Juli 2011	Kantonsschule, Willisau
August 2008 - Juni 2009	High School Exchange, Plato High School, USA
August 2011 - Juli 2013	Way Up Lehre als Elektronikerin EFZ bei Toradex AG, Horw
September 2013 - Juli 2016	Elektrotechnikstudim Bachelor of Science Vertiefung Automation & Embedded Systems Hochschule Luzern - Technik & Architektur, Horw
Juli 2015 - Januar 2016	Austauschsemester, Murdoch University, Perth, Australien
September 2016 - jetzt	Elektrotechnikstudium Master of Science Hochschule Luzern - Technik & Architektur, Horw

Berufliche Tätigkeit

Juli 2003 - August 2003	Produktionsarbeiten bei Schmidiger GmbH, Menzberg
Juli 2004 - August 2004	Produktionsarbeiten bei Schmidiger GmbH, Menzberg
Juli 2005 - August 2005	Produktionsarbeiten bei Schmidiger GmbH, Menzberg
Juli 2006 - August 2006	Produktionsarbeiten bei Schmidiger GmbH, Menzberg
Juli 2007 - August 2007	Produktionsarbeiten bei Schmidiger GmbH, Menzberg
Juli 2009 - August 2009	Produktionsarbeiten bei Schmidiger GmbH, Menzberg
Juli 2010 - August 2010	Produktionsarbeiten bei Schmidiger GmbH, Menzberg
Juli 2014 - August 2014	Schwimmlehrerin bei Matchpoint Sports Baleares, Mallorca
September 2016 - jetzt	Entwicklungsingenieurin bei EVTEC AG, Kriens