MASTER OF SCIENCE
IN ENGINEERING

Lucerne University of
Applied Sciences and Arts

HOCHSCHULE
LUZERN

Technik & Architektur
FH Zentralschweiz

# $\LaTeX$
# HSLU Elektrotechnik Master

## Template, Grundlagen, Tipps, Vorlagen

**Stefanie Schmidiger**

MASTER OF SCIENCE IN ENGINEERING
Vertiefungsmodul I

Advisor: Erich Styger

Experte: Der Experte

Horw, 2017

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, 10.01.2017 Stefanie Schmidiger

Versionen

| | | | |
|---|---|---|---|
| Version 0 | Vorabzug | 01.09.14 | Stefanie Schmidiger |

# Vorwort

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.
In this project, a hardware has been designed where multiple data inputs and outputs and multiple transmitters can be connected to a serial switch. The designed hardware features an SD card with a configuration file where data routing can be configured.
Data from connected devices will be collected and put into a data package with header, checksum, time stamp and other information. The package is then sent out via the configured transmitter. The corresponding second serial switch hardware receives this package, extracts and checks the payload, sends it out to the corresponding device and sends an acknowledge back to the package sender.
When data transmission over one transmission technology fails, the configuration file lets the user select the order of back up transmitters to be used. Data priority can also be configured because reliability of data transmission is extremely important with information such as exact location of the drone but not as important with information such as state of charge of the battery.
The serial switch hardware designed in the scope of this project features four serial RS232 connections where input and output devices can be connected that process or generate data. There are also four RS232 connectors where transmitters can be connected to send or receive data packages. The routing between data generating devices and transmitters to use can be done in a .ini file saved on an SD card.
There are two SPI to UART converters that act as the interface between the four devices connected and the micro controller respectively the four transmitters and the micro controller.
In a first version of the project, a Teensy 3.1 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 first version of the software ran with more than ten tasks and an overhaul of the complex structure was necessary.
For this reason, an adapter board has been designed so the existing hardware could be used with the more powerful Teensy 3.5. This adapter board features a SWD hardware debugging interface that was ready to use after removing a single component on the Teensy 3.5 development board.
The Teensy 3.5 was then configured to run with FreeRTOS. Task scheduler and queues provided by this operating system have been used to develop software that extracts data from received packages to output them on the configured interface or generates packages from received data bytes to send them out over the configured transmitter. The concept of acknowledges has also been applied so package loss can be detected and lost packages can be resent.
The software concept implemented 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 3.1 and Arduino, a refactoring has been necessary. Now further improvements and extra functionalities can be implemented more easily.

Horw, January 2017 Stefanie Schmidiger

# Kurzfassung

Hier wird der gesamte Text der Kurzfassung eingefügt.

# Inhaltsverzeichnis

# 1 Task Description

This project has been done for the company Aeroscout. 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 from Bild 1.1

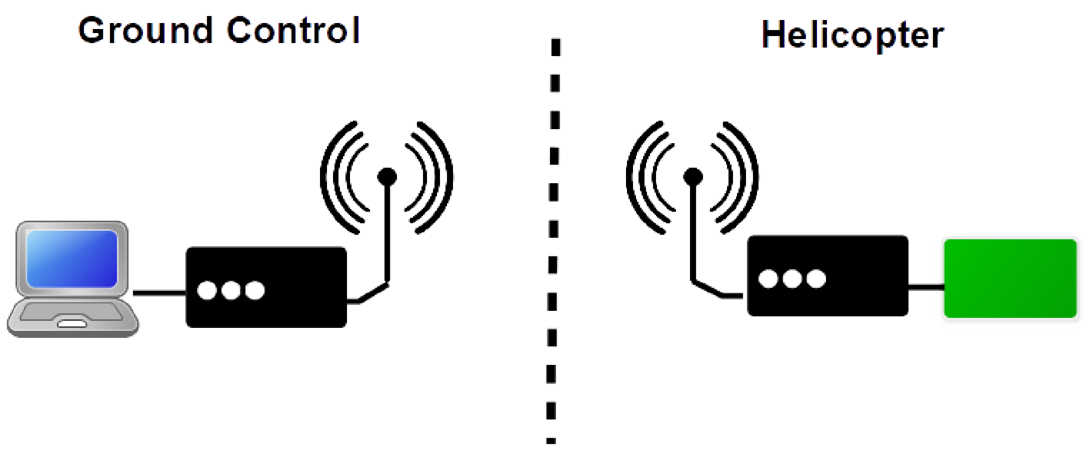


**Bild 1.1:** Previous system setup for data transmission

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, an other technology can be chosen 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 Bild 1.2

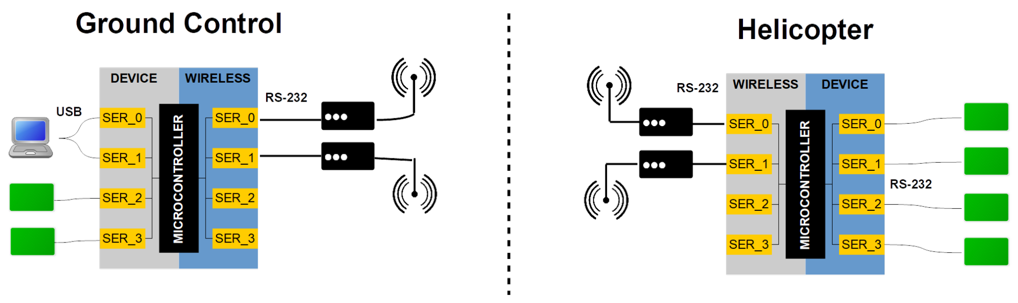


**Bild 1.2:** New system setup for data transmission

There are various kinds 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 interface in case sensitive data is exchanged over a connection.
The finished product should be have a 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.
A detailed description of all the requirements can be taken from the appendix Aufgabenstellung .

Link zur Aufgabenstellung im Appendix

# 2 Starting Situation

It was not necessary to start from scratch for this project.
In the beginning of 2017, Andreas Albisser has already started with an implementation and provided a first solution.
He developed a hardware that was used as the interface between input/output data and modem for wireless transmission. He chose the Teensy 3.1 development board as a micro controller and worked with the Arduino IDE and Arduino libraries.
There are various problems still with his work which lead to this follow up project to improve the overall functionality.
More details about the work Andras Albisser has done can be taken from this chapter.

## 2.1 Hardware

The hardware developed by Andreas Albisser has a total of eight interfaces where peripheral devices can be connected. Four connections are for control units, sensors or any other devices that process or generate data to be transmitted. On the other side, there are four connectors for modems to allow different ways of transmission. An overview can be seen in Bild 2.1.

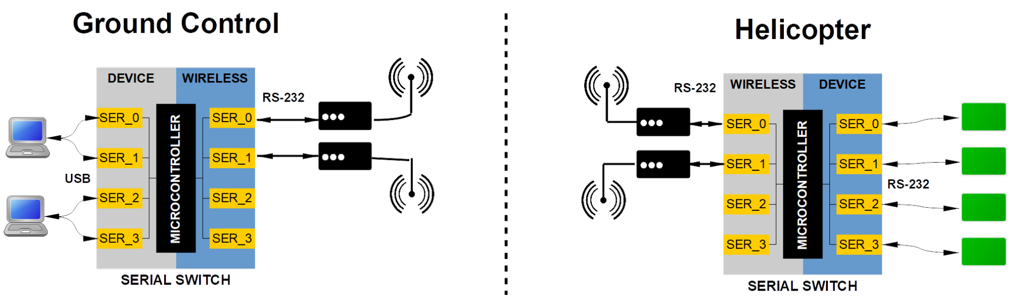


**Bild 2.1:** Hardware overview

Each interface accessible to the user is bidirectional which means that both sensors and actors can be connected.
From now on, the side where data generating and 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.
On both device side and wireless side, periphery can be connected to the four UART serial interfaces. On device side, the user can chose between a UART interface and a USB mini interface individually for each interface with jumpers. When selecting the USB mini interface, one USB hub acts as a dual COM interface, allowing two serial COM ports to open up to simulate two serial interfaces.
The serial interfaces are not connected to the Teensy 3.1 development board directly. There is a SPI to UART converter that acts as a hardware buffer between serial input/output and micro controller. All serial connections work on RS232 level which is +-12V. Because the SPI to UART converter is not

RS232 level compatible, a voltage regulator is used between the serial interface accessible to the user and the SPI to UART converter.
Details about the components used on this hardware can be taken from the following section. A block diagram of the on-board hardware components can be taken from Bild 2.2.

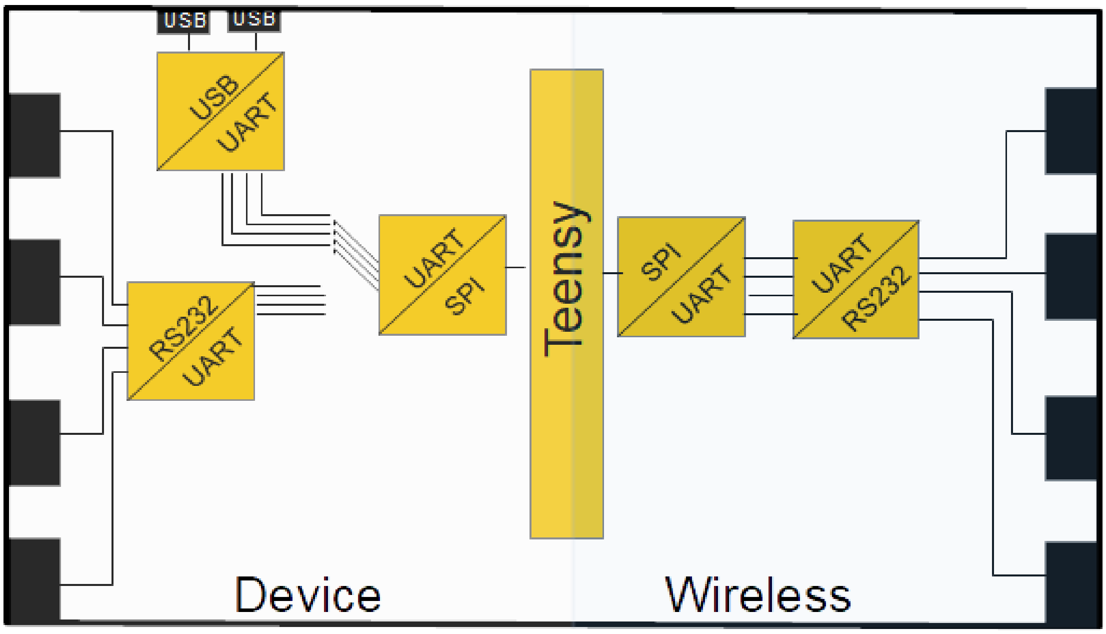


**Bild 2.2:** Hardware details

### 2.1.1 Serial Interfaces

There are a total of eight UART serial connections accessible to the user, four on device side and four on wireless side.
The baud rate for each serial connection can be configured individually.
UART is an asynchronous serial interface which means that there is no shared clock line between the two components. Both sides need to be configured with the same baud rate so they can communicate correctly.
A UART interface requires three wires: two unidirectional data lines (RX and TX) and a ground connection. Those three wires are accessible to the user, but with RS232 level, which is +-12V.

### 2.1.2 RS232 to UART Converter

The serial interfaces accessible to the user work on RS232 level. Just behind the serial interface, there is a level shifter that converts the RS232 level to TTL (5V).
This level shifter is bypassed on the device side in case the USB serial connection is used instead of the RS232 serial interface.

### 2.1.3 USB Interface

On device side, the user can chose whether data is provided via USB or via RS232 serial connection.
A jumper is used to switch between RS232 input and USB input.
In case when the USB input is selected, each USB hub acts as a dual serial COM port which means that when connecting the hardware to a computer, there will be two COM ports available per USB connection.
The on board USB to UART converter acts as an interface between USB hub and SPI to UART converter.

### 2.1.4 SPI to UART Converter

UART is an asynchronous serial interface which requires three connections: ground and two unidirectional data lines. If the teensy was to communicate to each serial port directly, it would require eight of those UART interfaces (which would add up to 16 data lines). To facilitate communication to the serial interfaces, a SPI to UART converter was selected as an intermediate interface.
There are two SPI to UART converters on board, one for the four device serial connections and one for the four wireless serial connections. SPI is a synchronous master-slave communication interface where the unidirectional data lines are shared amongst all participants. The only individual line between master and slave is the Slave Select line that determines, which slave is allowed to communicate to the master at a time.
Those converters are used as hardware buffers and can store up to 128 bytes.

### 2.1.5 Teensy 3.1 Development Board

Andreas Albisser used a Teensy 3.1 as a micro controller as can be seen in Bild 2.3.
The Teensy development boards are breadboard compatible USB development boards. They are small, low-priced and yet equipped with a powerful ARM processor.
The Teensy development boards all come with a pre-flashed bootloader to enable programming over USB. They use a less powerful processor as an interface to the developer to enable the use of Arduino libraries and the Arduino IDE.
There is no hardware debugging interface available to the user on the Teensy development boards. Programming is only possible via USB.

### 2.1.6 Power Supply

The hardware needs 5V as a power supply. This can be achieved by using any of the USB connections or via a dedicated power connector located on the board.

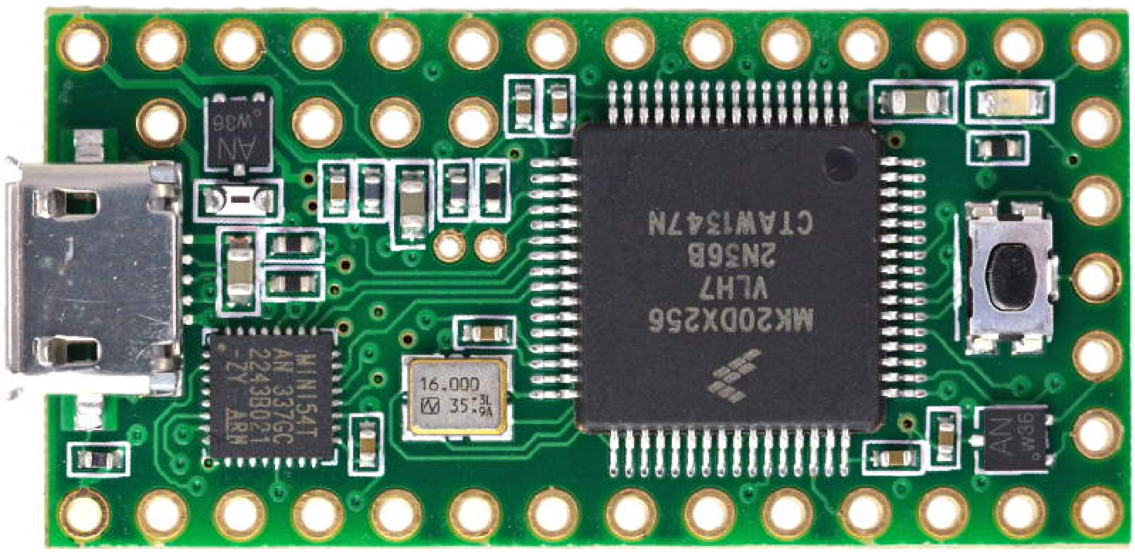


**Bild 2.3:** Teensy 3.1

## 2.2 Software

The following section gives you an overview of the Arduino software written by Andreas Albisser. There was only a brief documentation of the software available but fortunately, the comments in the code were helpful to get an understanding.
Any information provided below has been reverse engineered.

### 2.2.1 Software Development Tools

The software was written in C++ in Visual Studio. To compile the software, install the Visual Studio Enterprise 2015 version 14, the Arduino IDE extension for Visual Studio and the libraries "Queue by SMFSWänd "TaskScheduler by Anatoli Arkhipenko". Additionally, the old Arduino IDE version 1.8.1 has to be installed as well, together with the software add-on for Teensy support (Teensyduino). A detailed installation manual for all packages and environments needed can be found in the appendix.
.

Link zur Installationsanleitung im Appendix

### 2.2.2 Basic Functionality

The software written by Andreas Albisser provided a good basis and reference for the software developed in the scope of this project.
The basic functionality provided by his software was the transmission of data packages and acknowledges on wireless side. Generally it can be said that only packages are exchanged over wireless side and bytes are transmitted or received over device side.
The Teensy would frequently poll its SPI to UART hardware buffers for received data. In case the SPI to UART converter had data in its device buffer, the Teensy would read the data in a second SPI command. The read data is then wrapped in a package with header which contained CRC, timestamp and other information and sent out on the wireless side.
The corresponding second hardware would receive this package on its wireless side, extract the payload from it and send the extracted payload out on its device side.

To ensure successful transmission of packages, the concept of acknowledges is applied in the software where the receiver replies with an acknowledge to a successful package reception. A sequence diagram of a successful package transmission can be found in Bild 2.4.
Both Serial Switches continuously do both tasks: poll on device side to generate data packages for sending and poll on wireless side to receive wireless packages and send that payload back out on its device side.

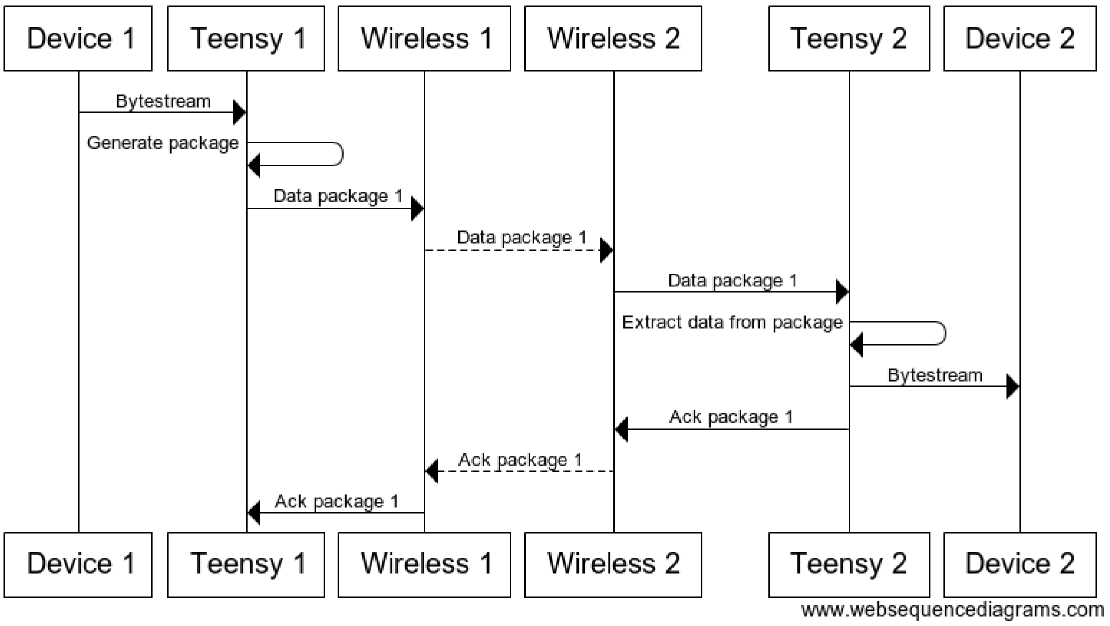


**Bild 2.4:** Successful package transmission

The maximum number of payload bytes per package can be configured in the software, just like the maximum amount of time the application should wait for a package to fill up until it will be sent anyway.
In case the package transmission was unsuccessful, either if the package got lost or corrupted, the receiving hardware will not send an acknowledge back. The application that sent the package will wait for a configurable amount of time before trying to send the same package again. Details can be found in figure Bild 2.5.
The maximum time to wait for an acknowledge before resending the same package can be configured in the software. The maximum number of resends per package can be configured for each wireless connection.

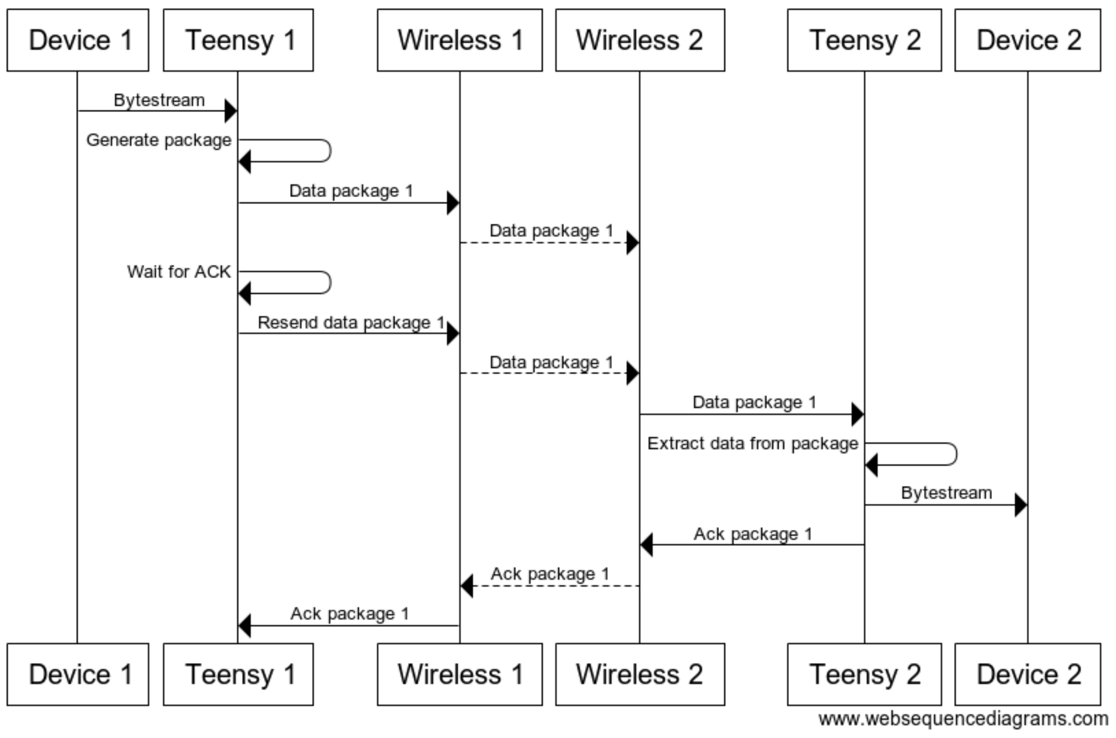


**Bild 2.5:** Unsuccessful package transmission

### 2.2.3 Configuration

All basic configuration parameters of the Arduino software are in the file serialSwitch_General.h For changes to be executed, the software has to be recompiled and uploaded. In order to do so, the necessary environment and all packages used by the software have to be installed on the computer as described in the user manual . All configuration possibilities of Andreas Albissers software can be taken from the Tabelle 2.1:

Link zum user manual FW installation von Andreas

| 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. |
| MAX_THROUGHPUT_WIRELESS_CONN | 0..4294967295 | Limit of the maximum data throughput in bytes/s per wireless connection. If two devices use the same wireless connection with the same priority but the maximum throughput is reached, data of the lower priority device will be redirected to its wireless connection with the next lower priority or discarded (in case this was the wireless connection with lowest priority already). Example: 0, 10000, 10000, 10000 means that wireless connection 0 will not be used. |

| | | |
|---|---|---|
| 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. |

**Tabelle 2.1:** Configuration parameters of arduino software

Further configuration parameters can be found in the file serialSwitch_General.h
There, the you can modify task interval of all tasks, enable hardware loopback and debug output or edit the preamble for a package start.

| **Configuration parameter** | **Possible values** | **Description** |
|---|---|---|
| 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. |
| ENABLE_TEST_DATA_GEN | 0, 1 | Random test data will be generated instead of waiting for device data to arrive to fill a package. |
| GENERATE_THROUGHPUT_OUTPUT | 0, 1 | Information about the data throughput on wireless side will be printed out on the serial terminal. |
| X_INTERVAL | 0...65535 | Task interval in milliseconds for each task. Replace X with the name of the task. |

**Tabelle 2.2:** General software configuration

### 2.2.4 LED Status

There is a separate task that handles blinking of the green LED. While this LED blinks, the software is running and all threads are executed.
The orange LED is turned on when a warning is printed out on the serial interface and the red LED is turned on when an error occurs.

### 2.2.5 Software Concept

The software written by Andreas Albisser runs with ten main tasks that make up the basic functionality and several minor task that are responsible for debug output, blinking of LEDs and any other functionalities that can be enabled through the configuration header. The software concept can be seen in Bild 2.6.

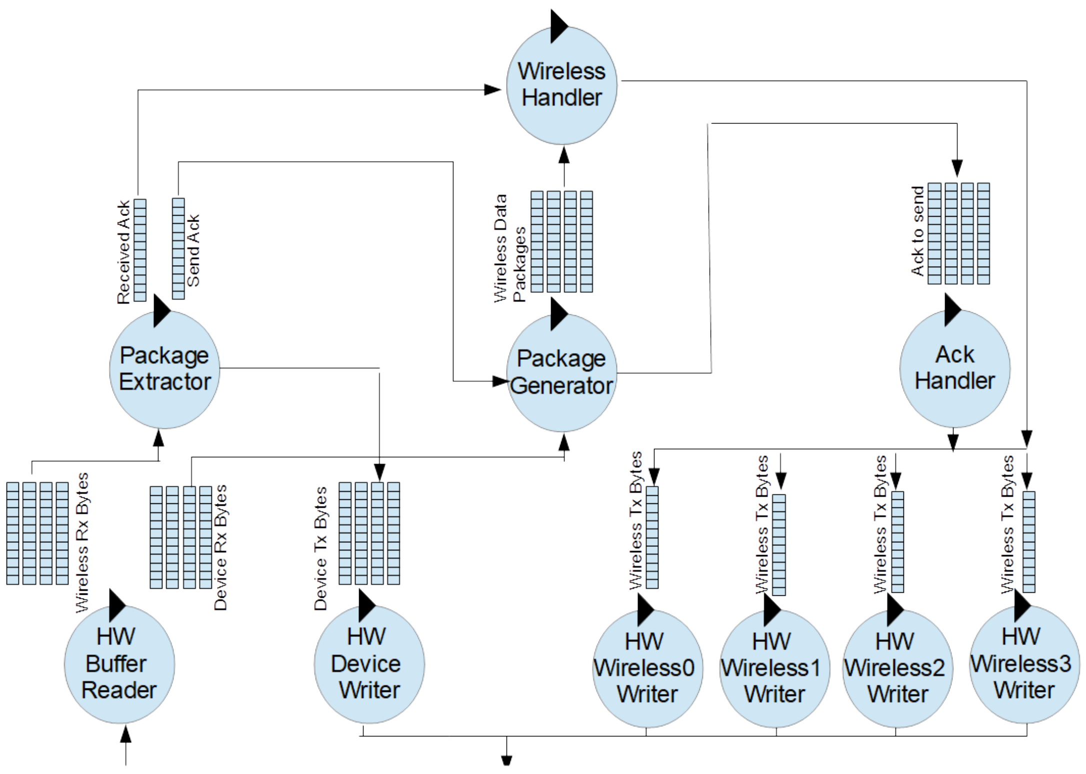


**Bild 2.6:** Arduino software concept

In the following sections, an overview will be given on the functionality performed by each task.

#### HW Buffer Reader

This task periodically polls the SPI to UART converters for new data. In case the converters have received data, the HW Buffer Reader will read and store the data in the corresponding queue.

The HW Buffer Reader does not know anything about packages or any data structure. It simply reads bytes and stores them in a queue.
The HW Buffer reader is responsible for the input data of both SPI to UART converters, the one on device side and on wireless side. This task has eight queues where the read data is stored, one queue for each UART interface accessible to the user.
If there is more data available in the hardware buffer (SPI to UART converter) than can be stored in the corresponding output queue of the HW Buffer Reader, the HW Buffer Reader will flush the queue to discard all information previously read from the SPI to UART converter and store its read data in the now empty queue.

```
/* send the read data to the corresponding queue */
/*const char* buf = (const char*) &buffer[0]; */
for (unsigned int cnt = 0; cnt < dataToRead; cnt++)
{
    if ((*queuePtr).push(&buffer[cnt]) == false)
    {
        /* queue is full - flush queue, send character again and set error */
        (*queuePtr).clean();
        (*queuePtr).push(&buffer[cnt]);
        if (spiSlave == MAX_14830_WIRELESS_SIDE)
        {
            char warnBuf[128];
            sprintf(warnBuf, "cleaning full queue on wireless side, UART number %u", (
                unsigned int)uartNr);
            showWarning(__FUNCTION__, warnBuf);
        }
        else
        {
            /* spiSlave == MAX_14830_DEVICE_SIDE */
            char warnBuf[128];
            sprintf(warnBuf, "cleaning full queue on device side, UART number %u", (
                unsigned int)uartNr);
            showWarning(__FUNCTION__, warnBuf);
        }
    }
}
```

### HW Device Writer

This task transmits data to the SPI to UART converter on device side. It takes bytes from its queues and passes them to the SPI to UART converter.
Communication to other tasks has been realized through four byte queues, one for each device interface accessible to the user.
This task does not know anything about data packages or other data structures, it only takes bytes from the queues and writes them to the SPI to UART converter on device side.

### HW WirelessX Writer

There are four tasks responsible for writing data to the SPI to UART converter on wireless side. Each task has its byte queue where data will be taken from and transmitted to the corresponding wireless user interface.
These tasks do not know anything about data packages or other data structures, they only take bytes from the queues and write them to the SPI to UART converter on wireless side.
Data is only taken from the queue and written to the hardware buffer if there is space available on the hardware buffer.

### Package Extractor

This task reads the wireless bytes from the output queue of the HW Buffer Reader and assembles them to wireless packages.
There are two types of wireless packages, acknowledges and data packages. The Package Extractor detects of which type an assembled package is and puts it on the corresponding queue.
This task also verifies the checksums of both header and payload of a package and discards the package in case of incorrect checksum.
In case of full output queues, new packages will be dropped and not stored in the corresponding queue.

### Package Generator

This task reads the incoming device bytes from the output queue of the HW Buffer Reader and generates data packages with this device data as payload. The generated packages are then stored in the corresponding queue for further processing.
The Package Generator also generates acknowledge packages when being told so by the output queue of the Package Extractor. The generated acknowledge are then put into the correct queue for a wireless connection.
If the queue is full, the package is dropped, no matter if acknowledge package or data package.

```
/* check if there are some receive acknowledge packages that needs to be created */
while (queueSendAck.pull(&ackData))
{
    if (generateRecAckPackage(&ackData, &wirelessAckPackage))
    {
        queueWirelessAckPack.push(&wirelessAckPackage);
    }
}

/* generate data packages and put those into the package queue */
if (generateDataPackage(0, &queueDeviceReceiveUart[0], &wirelessPackage))
{
    queueWirelessDataPackPerDev[0].push(&wirelessPackage);
}
if (generateDataPackage(1, &queueDeviceReceiveUart[1], &wirelessPackage))
{
    queueWirelessDataPackPerDev[1].push(&wirelessPackage);
}
if (generateDataPackage(2, &queueDeviceReceiveUart[2], &wirelessPackage))
{
    queueWirelessDataPackPerDev[2].push(&wirelessPackage);
}
if (generateDataPackage(3, &queueDeviceReceiveUart[3], &wirelessPackage))
{
    queueWirelessDataPackPerDev[3].push(&wirelessPackage);
}
```

The queue function call for push() returns false if unsuccessful there is no handling of an unsuccessful push in this code.

### Ack Handler

This task takes the acknowledge package generated by the Package Generator, splits it into bytes and puts those bytes into the corresponding wireless queue for the HW WirelessX Writer to send out.

**Wireless Handler**

This task handles the correct sending of wireless packages. It takes wireless packages from the output queues of the Package Generator, splits them into bytes and puts those bytes to the queue of the correct HW WirelessX Writer.
This task has an internal buffer where packages with an expected acknowledge are stored. The Wireless Handler keeps track of the send attempts per wireless connection and handles the resending of packages.
The Wireless Handler also does the prioritizing of data packages.

### 2.2.6 Wireless Package Structure

There are two types of packages that are exchanged over wireless side: acknowledges and data packages. Each package consists of a header and a payload of arbitrary size. Acknowledges and data packages can be distinguished by a parameter in the header of a package.
More information about the structure of header and payload can be found in the following section.

**Header**

The structure of a header can be found in Tabelle 2.3.

| Parameter name | Description | Value range | Length, bytes |
|---|---|---|---|
| PACK_START | Preamble for a package header start, indicates the beginning of a header. | 0x1B | 1 |
| PACK_TYPE | Determines weather it is a data package or acknowledge. | 1: data pack, 2: acknowledge | 1 |
| SESSION_NR | A random number generated upon startup of the software to keep the receiver from discarding all packages in case a reset has been made. | 0...255 | 1 |
| SYS_TIME | Milliseconds since start of the software. This parameter is used as a substitute for package numbering. | 0...4294967295 | 4 |
| PAYLOAD_SIZE | Number of bytes in payload of this package | 0...65535 | 2 |
| CRC8_HEADER | 8 bit CRC of this header | 0...255 | 1 |

**Tabelle 2.3:** Header structure

When the software receives a package, it checks the system time to decide if it should be discarded. If the system time of a received package is lower than the last one received, it will be discarded.
In case of a hardware reset, the system time starts over with 0 which means that all packages would be discarded on receiving side. This is the reason for the session number. When the session number changes, the receiver knows to start over with the system time and not to discard all received packages.

**Payload**

Talk about payload and CRC32

| Parameter name | Description | Value range | Length, bytes |
|---|---|---|---|
| PAYLOAD | Data bytes to send out | 0...255 | 0...65535 |
| CRC16_PAYLOAD | 16 bit CRC of the payload | 0...65535 | 2 |

**Tabelle 2.4:** Payload structure

## 2.3 Discussion and Problems

There was no documentation available on tests conducted or issues discovered with Andreas Albissers work.
All information below has been recalled by the people involved.

### 2.3.1 Tests Results

The software was only tested briefly and testing was not documented.
The end test consisted of

Testing with modem and link explained

#### Directly connected wireless sides

When connecting two serial switches directly with a cable, the functionality was as expected. Acknowledges were transmitted and packages resent in case no acknowledge was received. Packages were not acknowledged when the checksum did not match and were resent after the configured delay.

#### Wireless sides with modem

Problems arose when working with modems instead of a direct wired connection. When connecting one modem and one device only, the data stream was still reliable. But once two modems were connected and more than one device was generating data, the connection could not be established and no data stream was output on the receiving side.
Those tests were repeated when no acknowledge was configured but the result was similar: no data link could be established.
One possible reason for this faulty behavior could be the lack of package numbering. The header only contains session number and system time but no variable with a monotonic counter. The system time does not provide any information about missing packages because it might jump up in case no package was generated for some amount of time. The parameter system time corresponds to the runtime of the software since start up in milliseconds.
The following code section is copied from the software Andreas Albisser developed. It shows that packages will be discarded when their time stamp (sysTime) is older than the one of the last package.

```
/* make sure to not send old data to the device - but also make sure overflow is handled
    */
if ((currentWirelessPackage[wirelessConnNr].sysTime > timestampLastValidPackage[
    currentWirelessPackage[wirelessConnNr].devNum]) ||
((timestampLastValidPackage[currentWirelessPackage[wirelessConnNr].devNum] -
    currentWirelessPackage[wirelessConnNr].sysTime) > (UINT32_MAX / 2)))
{
    /* package OK, send it to device */
    timestampLastValidPackage[currentWirelessPackage[wirelessConnNr].devNum] =
        currentWirelessPackage[wirelessConnNr].sysTime;
    Queue* sendQueue;
    sendQueue = &queueDeviceTransmitUart[currentWirelessPackage[wirelessConnNr].devNum];
    for (uint16_t cnt = 0; cnt < currentWirelessPackage[wirelessConnNr].payloadSize; cnt
        ++)
    {
        if (sendQueue->push(&data[wirelessConnNr][cnt]) == false)
        {
            char warnBuf[128];
            sprintf(warnBuf, "Unable to send data to device number %u, queue is full",
                currentWirelessPackage[wirelessConnNr].devNum);
            showWarning(__FUNCTION__, warnBuf);
            break;
        }
    }
}
else
{
    /* received old package */
    /* also can happen when we have two redundant streams.. */
    static char infoBuf[128];
    sprintf(infoBuf,
    "received old package, device %u - check configuration if this message occurres often"
        ,
    currentWirelessPackage[wirelessConnNr].devNum);
    showInfo(__FUNCTION__, infoBuf);
}
```

### 2.3.2 Issues

#### Dropping Data when Transmission Unreliable

Task intercommunication has been done with queues, as can be seen in Bild 2.6. When data arrives, it will be pushed to the byte queue from one task to be processed and assembled to a full package by an other task.
When data arrives too fast, any of the queues may be full and pushing the most recent data to the queue might fail. Currently, only the hardware interfaces (HW Device Writer and HW WirelessX Writer) deal with this scenario. When they read data from the SPI to UART converter but can not push it to the queue, they will flush the queue (which results in all unprocessed data bytes being lost) and push the new data to the now empty queue. A warning will be printed on the serial interface afterwards.
A snippet from the code handling a full byte queue can be seen below.

```
if ((*queuePtr).push(&buffer[cnt]) == false)
{
    /* queue is full - flush queue, send character again and set error */
    (*queuePtr).clean();
    (*queuePtr).push(&buffer[cnt]);
```

When any of the other queues are full or pushing new data to the queue is not possible, this data will be lost and no further action will be taken except for printing a warning on the serial interface.
This results in packages being lost when lots of data is read on device side and packages generated. The package generator does not check if its package output queue is full before popping bytes from the received byte queue and putting them into the payload of a generated package. The generated package will be lost together with all its data when trying to push it onto a full queue.
All tasks should take the state of their queues during runtime into account so that dropping of data can be carried out controlled and purposely.

**Debugging**

Arduino does not support hardware debugging, only prints on a serial connection are available.
This is fine as long as the software is relatively small and straightforward.
As software complexity increases it becomes problematic, especially when multiple tasks are involved.
As can be seen in Bild 2.6, the software concept implemented by Andreas Albisser is complex and runs with many tasks. In order to expand the functionality of it even further, a real hardware debugging interface is inevitable.

**Software Concept**

The software concept implemented by Andreas Albisser as can be seen in Bild 2.6 and it was attempted to visualize it split into three layers similar to the ISO/OSI model.
All hardware reader and writer tasks access the SPI to UART interface and represent layer 1. They deal with bytes and do not know anything about packages.
Each wireless interface runs with its own task that handles the bytes to be sent out. There is one task that writes bytes to the SPI to UART converter on device side and one task that reads incoming bytes from both SPI to UART converters. These five tasks all access the same SPI interface which results in possible conflicts and the need to define critical sections. The software concept would be simpler if there was only one task that accessed the SPI interface.
The package generator, package extractor and ack handler tasks are the interface between bytes and packages and represent layer 2. They assemble wireless packages popped from the output queue of layer 1 and split generated packages into bytes. They also deal with the sending of acknowledges in case a received package expects one.
The package handlers have two separate queues where assembled data packages and acknowledges are stored. The wireless handler will then not process packages in the order they were received but will first iterate through the data package queue and then through the acknowledge queue.
The wireless handler represents layer 3 and deals with sending and resending of packages. While it keeps track of acknowledges received, it does not handle the acknowledges sent because this is contradictory done by layer 2.
Generally, the software could be kept much smaller with less tasks and less queues.

**Data Priority**

There is no way to prioritize data of one device over data of an other device connected. When data transmission becomes unreliable, the software needs to know which data is most important to be exchanged. An additional configuration parameter should be added to represent data priority.

# 3 Hardware

The task description for this project as seen in requires the following hardware changes:

Auf-gaben-stel-lung

- Optimization of size and weight
- Optimization for outdoor use
- Usage of more powerful processor with more memory and RNG or encryption support
- SWD/JTAG debugging interface
- UART hardware flow control
- On-board SD card (regular or micro)

There are several options on how to proceed with the implementation. The pro and cons of these choices are listed in this chapter, followed by the chosen solution and its execution.

## 3.1 Hardware Redesign Options

The hardware Andreas Albisser designed is working as expected. The desired modifications as listed in the task description are merely optimizations. Only the replacement of the Teensy 3.1 is absolutely required for this project because software will later be written for a specific micro controller.
Therefore there are two options on how to proceed.

- Redesign entire hardware for/with a new processor.
- Redesign hardware step by step, starting with just an adapter board to use existing hardware with new micro controller.

### 3.1.1 Complete Hardware Redesign

A redesign of the entire hardware requires careful component selection, adaption of the schematic and footprints and redesign of the PCB.
Changes for the next complete hardware redesign include:

- New RS232 level to TTL level converter with more inputs to convert hardware flow control pins (CTS/RTS) as well.
- A way to switch between RS232 level and TTL level for all serial interfaces accessible to the user.
- More powerful micro controller with support for encryption.
- SD card slot.
- Hardware debugging interface.
- Connectors for all serial interfaces with more pins to include hardware flow control.

Implementing all these changes would require about three weeks, followed by one week of manufacturing and one week of assembly. In case of a faulty hardware, it would be extremely difficult start with the software implementation because there is no hardware to work with and no way to work

standalone. In this case, producing a second version of the hardware would take up a considerable amount of time because of manufacturing time, assembly and testing.
There is simply not enough time to redesign the entire hardware in the scope of this project.
A possible project plan for this scenario can be found in Bild 3.1:

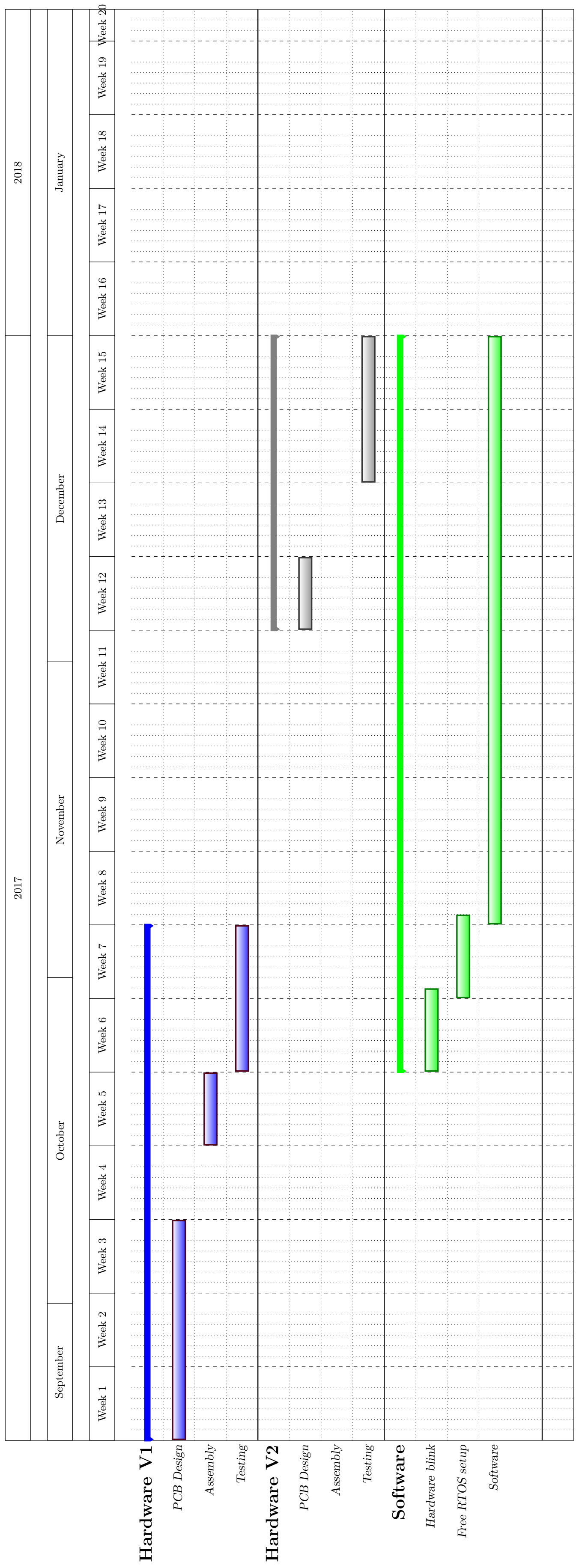


**Bild 3.1:** Possible project plan of a complete hardware redesign

### 3.1.2 Adapter Board

The most profound hardware change is the replacement of the Teensy 3.1.
First, a new development board or micro controller has to be selected that supports hardware debugging and meets the requirements for data encryption.
After selecting a replacement for the Teensy 3.1, the fastest way to get started with software development for the new micro controller is by designing an adapter board for the Teensy 3.1 footprint.
The Teensy 3.1 ships with headers that can be soldered onto the development board. Andreas Albisser designed the interface for the Teensy with female header pins so the development board could simply be plugged into the header and exchanged if needed. This makes it easy to design an adapter print with male headers that match the footprint of the previously used Teensy 3.1.
This solution has been chosen in the scope of this work to ensure that the end result of this project would provide solid ground work for further development.

## 3.2 Component Evaluation

Before starting with the design of an adapter board, a replacement for the Teensy 3.1 development board has to be chosen.

### 3.2.1 Development Board Selection

The easiest option is to select a more powerful Teensy development board that meets the requirements listed in .

Link zu Aufgabenstellung

Fortunately, both the Teensy 3.5 and Teensy 3.6 meet the requirements and have an on-board SD card slot. A comparison between the Teensies can be found in Tabelle 3.1 The pins of both Teensy 3.5 and

| | **Teensy 3.1** | **Teensy 3.5** | **Teensy 3.6** |
|---|---|---|---|
| **Processor** | MK20DX256<br>32 bit ARM<br>Cortex-M4<br>72 MHz | MK64FX512VMD12<br>Cortex-M4F<br>120 MHz | MK66FX1M0VMD18<br>Cortex-M4F<br>180 MHz |
| **Flash Memory [bytes]** | 262 k | 512 k | 1024 k |
| **RAM Memory [bytes]** | 65 k | 196 k | 256 k |
| **EEPROM [bytes]** | 2048 | 4096 | 4096 |
| **I/O** | 34, 3.3V, 5V tol | 58, 3.3V, 5V tol | 58, 3.3V, 5V tol |
| **Analog In** | 21 | 27 | 25 |
| **PWM** | 12 | 17 | 19 |
| **UART,I2C,SPI** | 3 | 6 | 6 |
| **SD Card** | no | yes | yes |
| **Price** | $19.80 | $25.25 | $29.25 |

**Tabelle 3.1:** Teensy comparison

3.6 are backwards compatible to the pin out of Teensy 3.2 which will make it easier to develop the PCB of an adapter board.
The Teensy 3.5 and Teensy 3.6 development board have all pins needed for SWD hardware debugging

available as pads on their backside.
The Teensy 3.5 was chosen for this application because there is more support available for this component and a FreeRTOS that is already configured. This is not the case with the Teensy 3.6.

### 3.2.2 Preparation for Hardware Debugging

The Teensy development boards are meant for USB programming and debugging. They are equipped with a small micro controller that acts as a boot loader. The small micro controller is in control of the hardware debugging and reset pins of the main micro controller and does the programming of the main micro controller. This way, all Teensies can be used with standard Arduino libraries and programmed with the Arduino IDE.
The schematic of the Teensy 3.5 can be seen in Bild 3.2. The MKL02Z32VFG4 acts as the boot loader and the MK64FX512 is the main micro controller.
The pins available to the user are shown in Bild 3.3 and Bild 3.4.

#### Serial Wire Debug

As can be seen in Bild 3.4, there are SWD (Serial Wire Debug) pins are available as pads on the back side of the Teensy 3.5.
The pinout of a SWD interface can be seen in Bild 3.5. Only the reset, data, clock and ground pins are absolutely required to be connected.
Even though those SWD pins are available on the backside of both Teensy 3.5 and Teensy 3.6, they are controlled by the boot loader.
There are two ways to communicate to the main micro controller directly without the boot loader interfering on the hardware debugging interface:

- Holding the boot loader in reset mode
- Removing the boot loader

#### Resetting the Boot Loader

According to the data sheet of the boot loader (see Bild 3.6 ), pin 15 can have one of three functions:

- Reset
- GPIO input
- GPIO output

As default, the pin will be configured as a reset pin, but this function can be turned off by configuring to any of the other two functions in software.
Even though the Teensies are fully open source, the software for the boot loader is not available. The only way to find out if the reset pin is still configured as such is by pulling it low and attempting a reset.
Before putting the boot loader into reset mode, any other functions that it may be responsible for need to be ensured.
Because the internal pull ups of the boot loader are used for the reset line of the main micro controller, this reset line needs to be pulled up externally first.
A resistor can be soldered onto the Teensy directly for this purpose as seen in Bild 3.7. Afterwards, pin 15 of the boot loader can be pulled low.

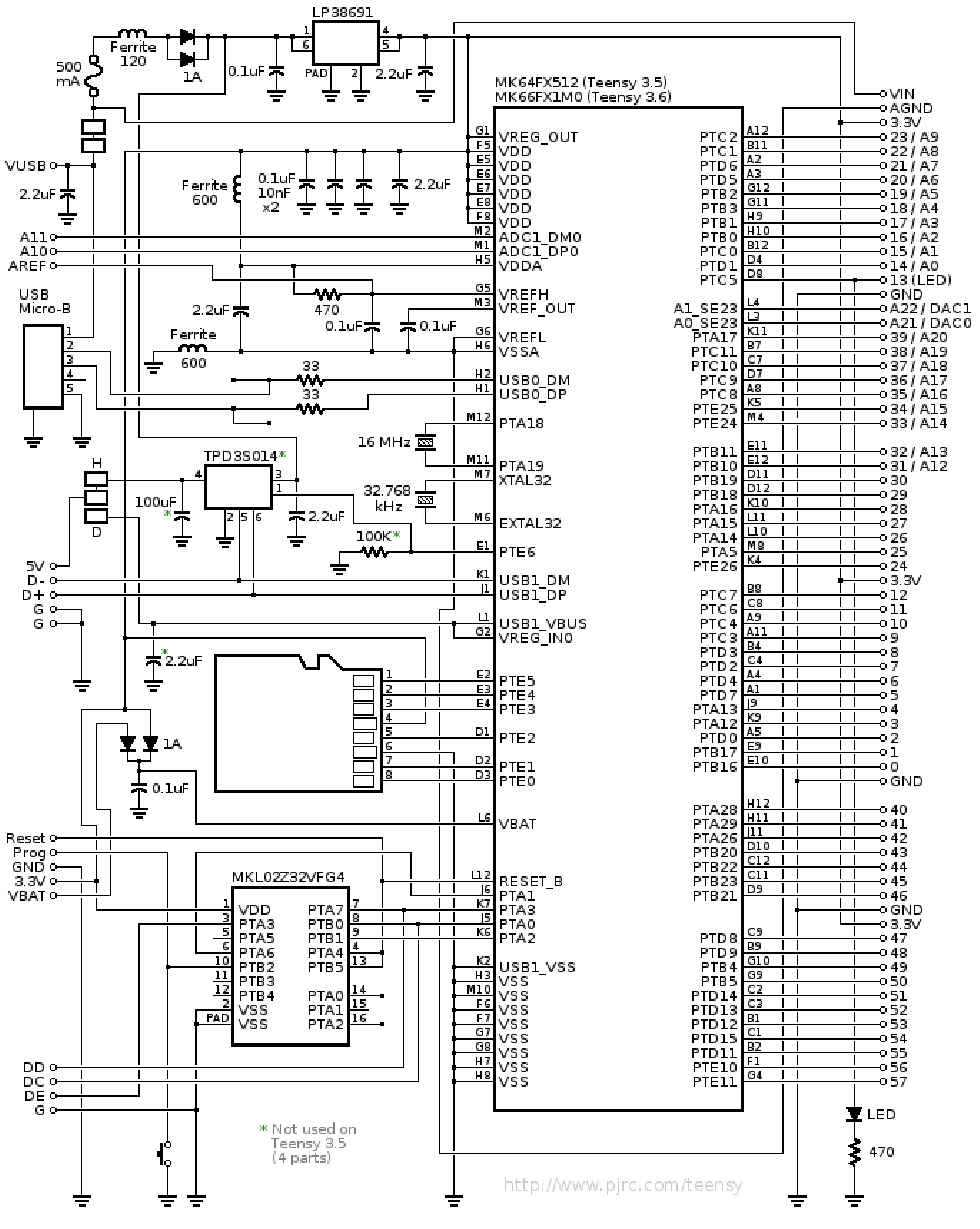


**Bild 3.2:** Schematic Teensy 3.5

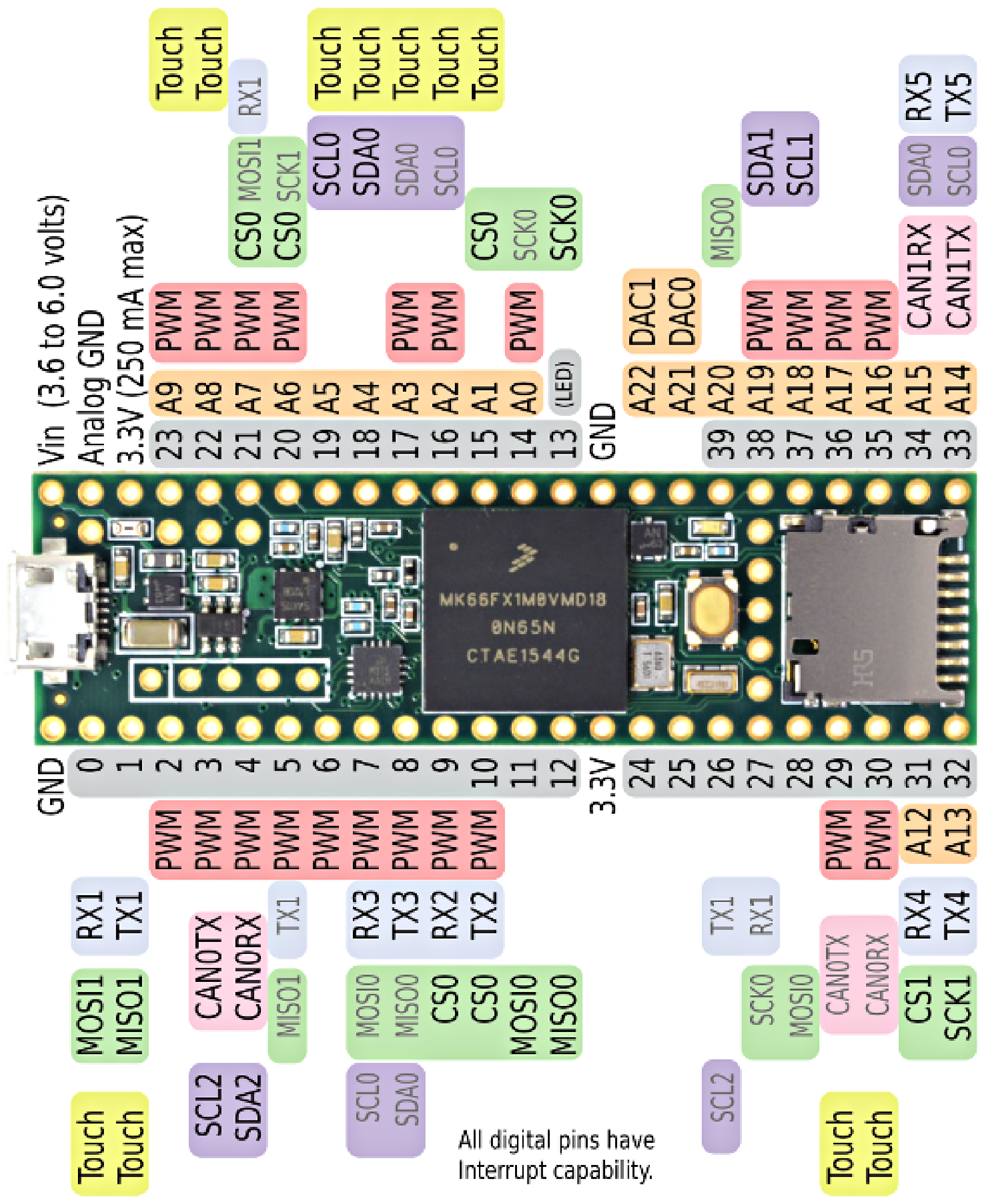


**Bild 3.3:** Pin assignment, front side

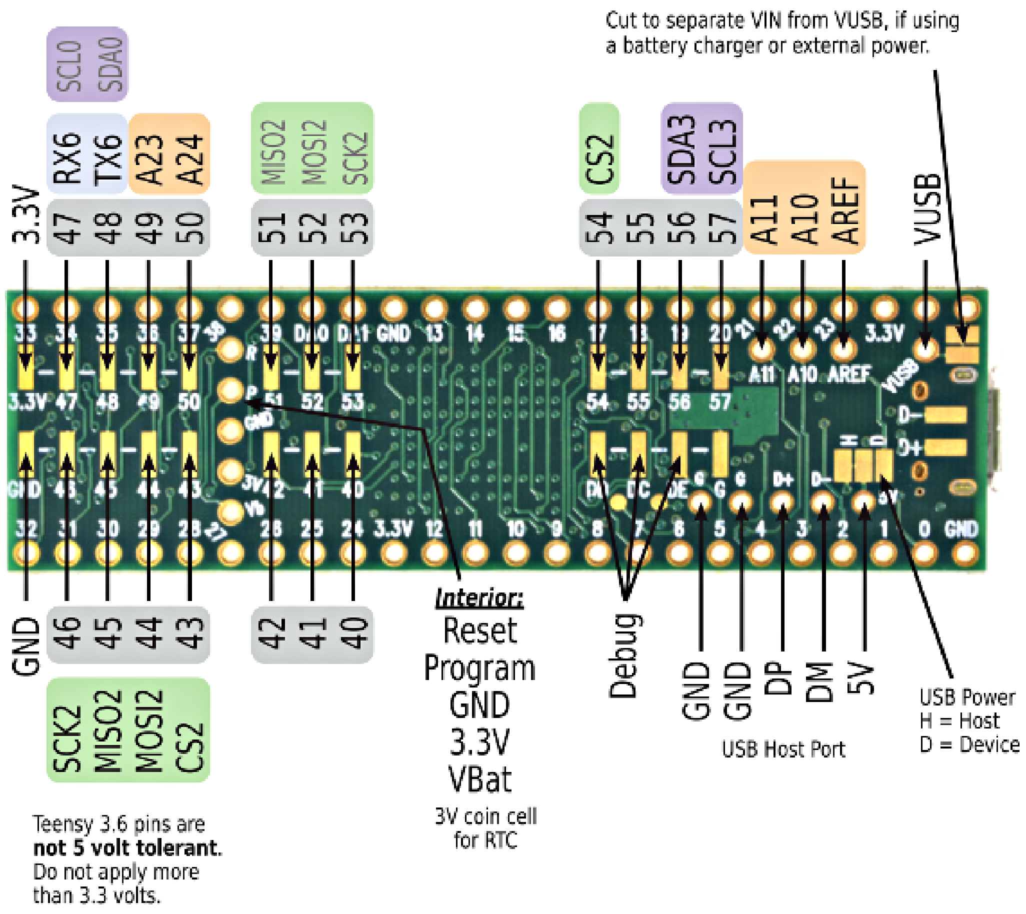


**Bild 3.4:** Pin assignment, back side

Unfortunately, pin 15 seems to be configured as a GPIO pin because pulling the boot loaders reset line low does not prevent it from communicating to the Teensy 3.5.
The state of the hardware debugging pins in idle state were checked with the scope but as can be seen from Bild 3.8, the boot loader is still in control of the debugging interface and therefore the main micro controller.
Instead of investigating further into this option, the second option was chosen.

### Removing the Boot Loader

The MKL02Z32VFG4 has two functions:

- It acts as a boot loader and controls the SWD interface to the main micro controller.
- Its internal pull ups are used to keep the main micro controllers reset line high.

To leave the user in full control of the SWD hardware debugging interface, the boot loader has to be removed (or silenced, as attempted in 3.2.2).
Flux gel was applied around the boot loader before heating the soldering pads up and removing the MKL02Z32VFG4. Now a pull up resister was added as seen in **??**. Afterwards, the SWD debugging interface could be used.

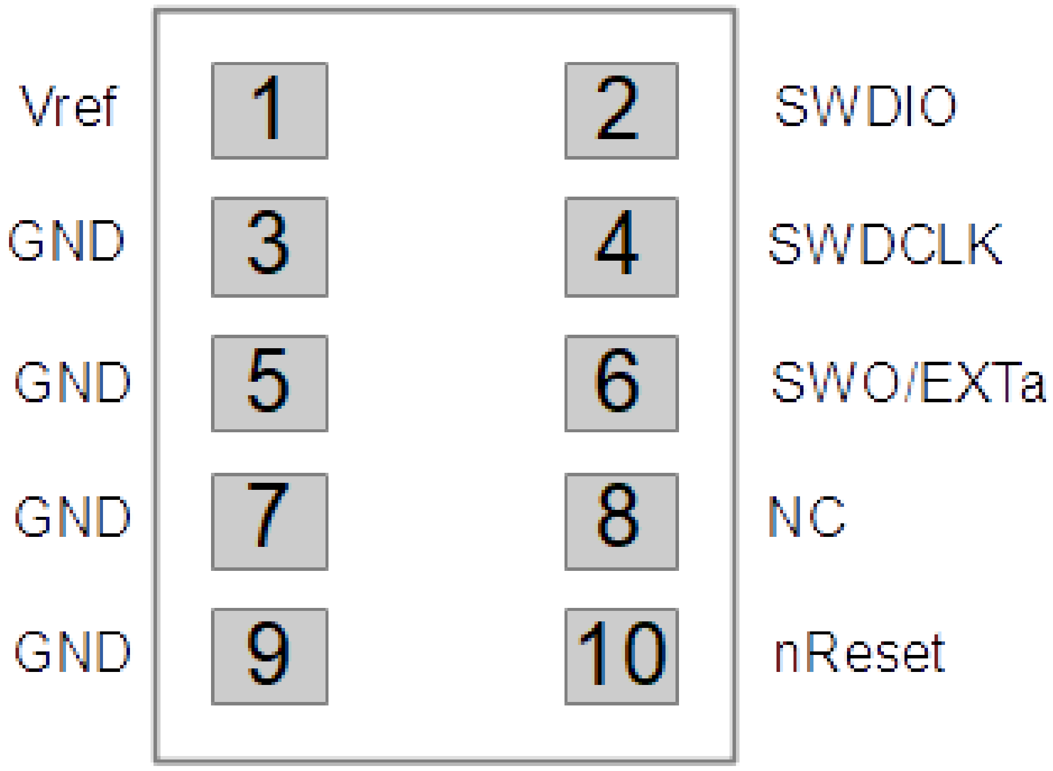
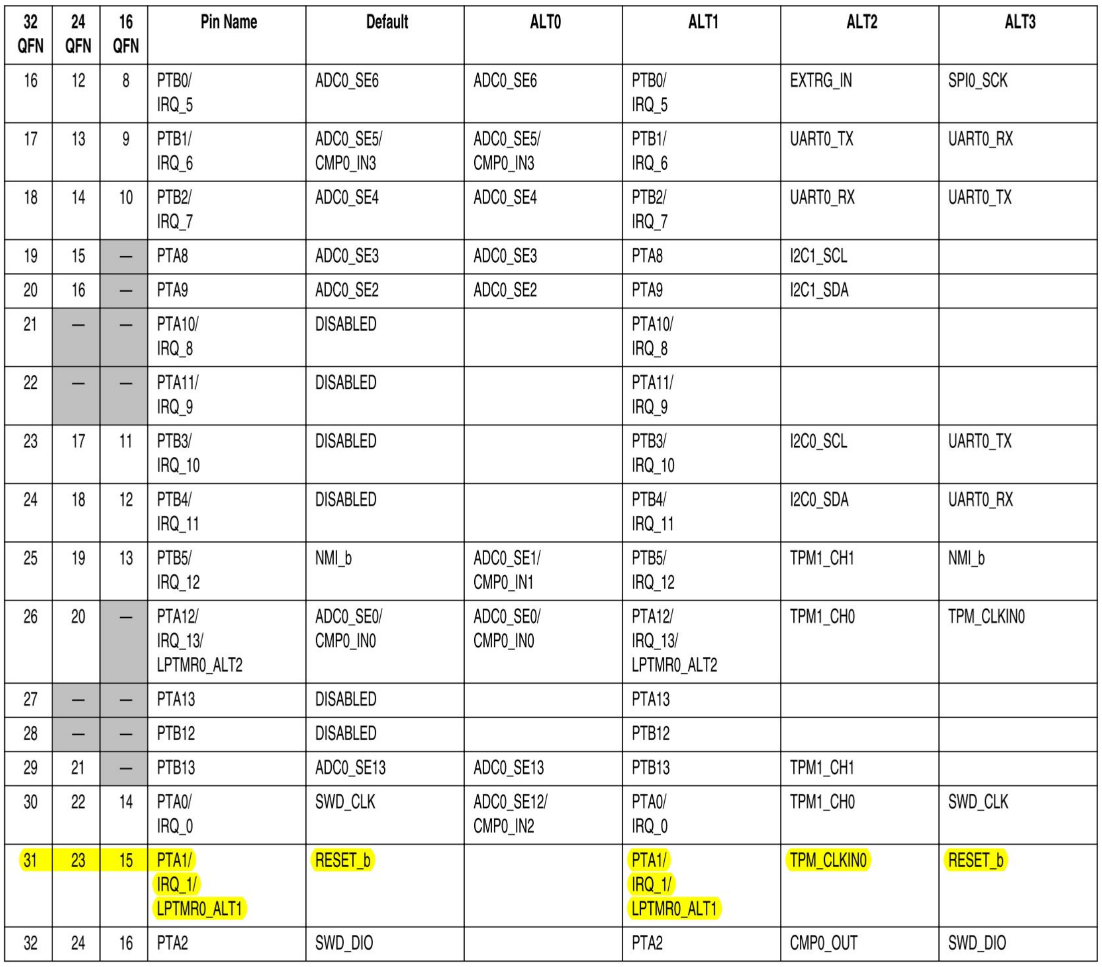


**Bild 3.5:** SWD pinout



| 32 QFN | 24 QFN | 16 QFN | Pin Name | Default | ALT0 | ALT1 | ALT2 | ALT3 |
|---|---|---|---|---|---|---|---|---|
| 16 | 12 | 8 | PTB0/ IRQ_5 | ADC0_SE6 | ADC0_SE6 | PTB0/ IRQ_5 | EXTRG_IN | SPI0_SCK |
| 17 | 13 | 9 | PTB1/ IRQ_6 | ADC0_SE5/ CMP0_IN3 | ADC0_SE5/ CMP0_IN3 | PTB1/ IRQ_6 | UART0_TX | UART0_RX |
| 18 | 14 | 10 | PTB2/ IRQ_7 | ADC0_SE4 | ADC0_SE4 | PTB2/ IRQ_7 | UART0_RX | UART0_TX |
| 19 | 15 | — | PTA8 | ADC0_SE3 | ADC0_SE3 | PTA8 | I2C1_SCL | |
| 20 | 16 | — | PTA9 | ADC0_SE2 | ADC0_SE2 | PTA9 | I2C1_SDA | |
| 21 | — | — | PTA10/ IRQ_8 | DISABLED | | PTA10/ IRQ_8 | | |
| 22 | — | — | PTA11/ IRQ_9 | DISABLED | | PTA11/ IRQ_9 | | |
| 23 | 17 | 11 | PTB3/ IRQ_10 | DISABLED | | PTB3/ IRQ_10 | I2C0_SCL | UART0_TX |
| 24 | 18 | 12 | PTB4/ IRQ_11 | DISABLED | | PTB4/ IRQ_11 | I2C0_SDA | UART0_RX |
| 25 | 19 | 13 | PTB5/ IRQ_12 | NMI_b | ADC0_SE1/ CMP0_IN1 | PTB5/ IRQ_12 | TPM1_CH1 | NMI_b |
| 26 | 20 | — | PTA12/ IRQ_13/ LPTMR0_ALT2 | ADC0_SE0/ CMP0_IN0 | ADC0_SE0/ CMP0_IN0 | PTA12/ IRQ_13/ LPTMR0_ALT2 | TPM1_CH0 | TPM_CLKIN0 |
| 27 | — | — | PTA13 | DISABLED | | PTA13 | | |
| 28 | — | — | PTB12 | DISABLED | | PTB12 | | |
| 29 | 21 | — | PTB13 | ADC0_SE13 | ADC0_SE13 | PTB13 | TPM1_CH1 | |
| 30 | 22 | 14 | PTA0/ IRQ_0 | SWD_CLK | ADC0_SE12/ CMP0_IN2 | PTA0/ IRQ_0 | TPM1_CH0 | SWD_CLK |
| 31 | 23 | 15 | PTA1/ IRQ_1/ LPTMR0_ALT1 | RESET_b | | PTA1/ IRQ_1/ LPTMR0_ALT1 | TPM_CLKIN0 | RESET_b |
| 32 | 24 | 16 | PTA2 | SWD_DIO | | PTA2 | CMP0_OUT | SWD_DIO |

**Bild 3.6:** Pin out MKL02Z32VFG4

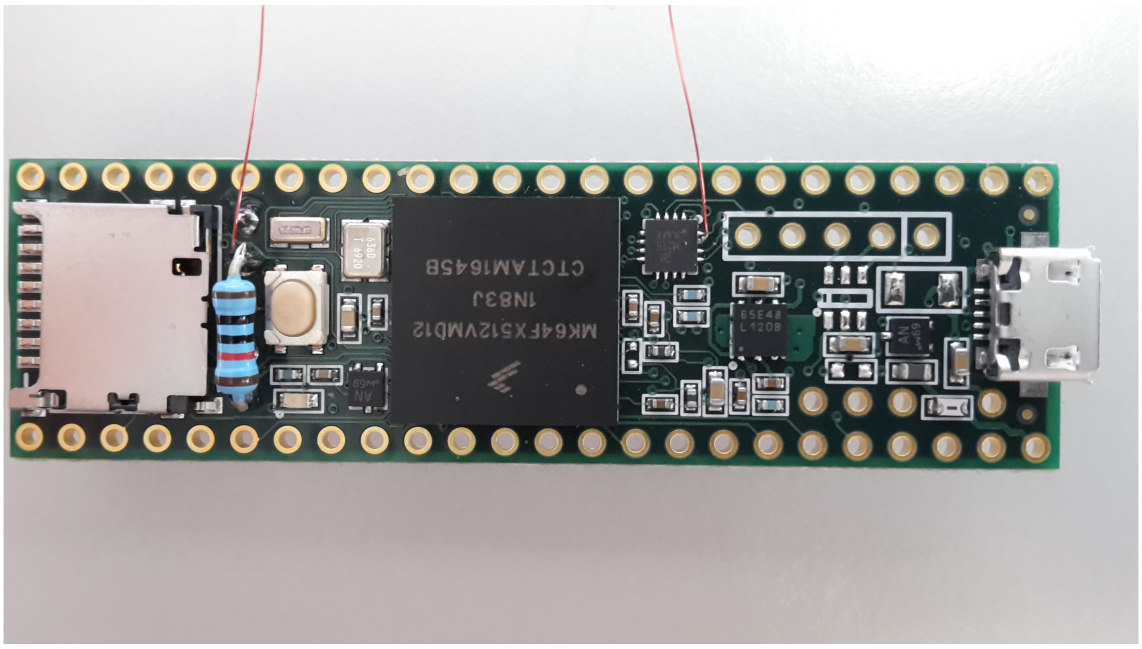


**Bild 3.7:** Trying to pull the boot loader into reset mode

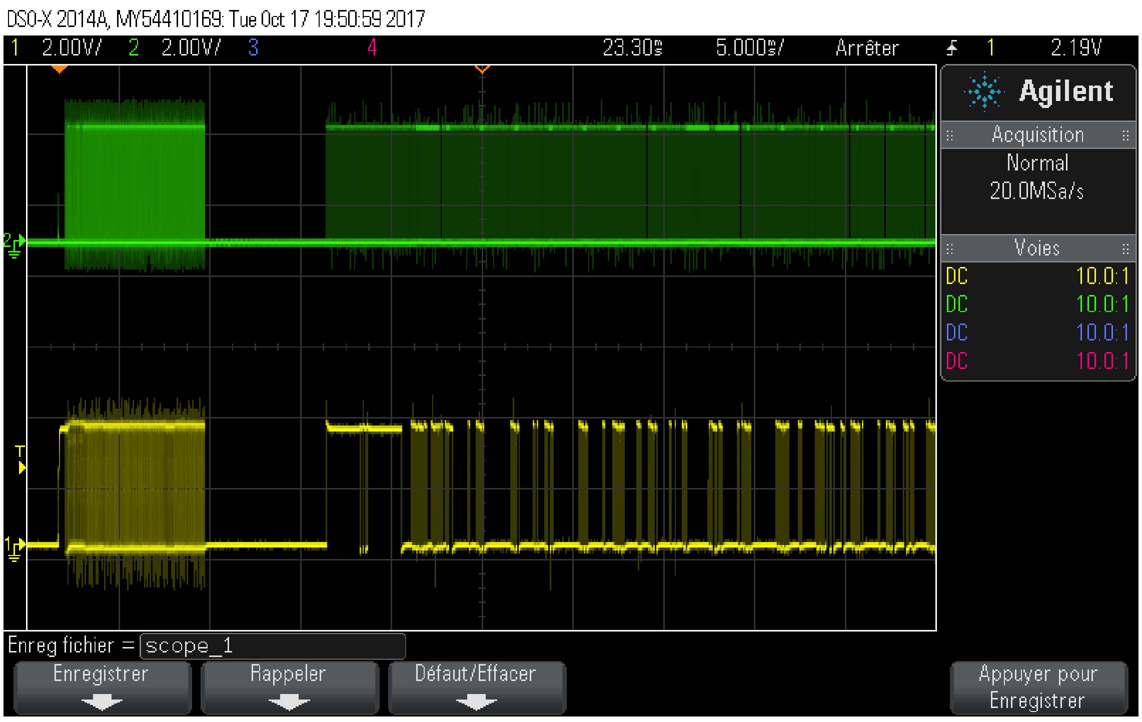


**Bild 3.8:** The boot loader keeps communicating to the Teensy

## 3.3 To-Do List for next version of UAV serial switch

List all the changes that are needed on the UAV base board
HW Flow control can’t be done without changing HW because lines are RS232 level!

# 4 Software

A complete documentation of the software written by Andreas Albisser for the Teensy 3.1 can be found in 2.2.
Various issues found with his software can be found in 2.3.
Now there are two options on how to proceed

- Port the existing software to C with FreeRTOS.
- Create a new software concept and implement it.

Both options are evaluated below.

### 4.0.1 Start with existing Software

## 4.1 New Software

Talk about new SW concept

#### To-Do List for new SW

Talk about the things that have not yet been implemented, such as:

- Package numbering instead of waiting for ACK
- Limit throughput
- No sending out the same package over multiple connections at the same time and only taking the one that was received first.
- Delay until package is dismissed -> this parameter is defined by timeout and maximum number of retries per connection anyway

# 5 Testing

Testing can be devided into multiple sections:

- Testing of hardware
- Testing of software
- Testing of functionalities with a modem attached

More details about the tests conducted on these items can be found below.

## 5.1 Hardware Tests

Testing of the core hardware was not done in the scope of this project because it was provided by Andreas Albisser.
Only the Teensy adapter board was tested.
Talk about the wrong footprint used.

## 5.2 Software Tests

In order to prevent memory leaks and package dropping, the software was tested as well:

- Echo
- Connecting two switches directly, two consoles used
-

## 5.3 Functionality Tests with Modem

blabla

### 5.3.1 Test Concept

-

## 5.4 System View

Activating system view will result in performance loss.
The system view logs events like queue calls and that logging needs to be deactivated in order to get a representative result.
Deactivate them by commenting the respective defines out in SEGGER_SYSTEM_VIEWER_FreeRTOS.h:

```
//#define traceQUEUE_PEEK( pxQueue )
    SYSVIEW_RecordU32x4(apiID_OFFSET + apiID_XQUEUEGENERICRECEIVE,
    SEGGER_SYSVIEW_ShrinkId((U32)pxQueue), SEGGER_SYSVIEW_ShrinkId((U32)pvBuffer),
    xTicksToWait, xJustPeeking)
//#define traceQUEUE_PEEK_FROM_ISR( pxQueue )
    SEGGER_SYSVIEW_RecordU32x2(apiID_OFFSET + apiID_XQUEUEPEEKFROMISR,
    SEGGER_SYSVIEW_ShrinkId((U32)pxQueue), SEGGER_SYSVIEW_ShrinkId((U32)pvBuffer))
//#define traceQUEUE_PEEK_FROM_ISR_FAILED( pxQueue )
    SEGGER_SYSVIEW_RecordU32x2(apiID_OFFSET + apiID_XQUEUEPEEKFROMISR,
    SEGGER_SYSVIEW_ShrinkId((U32)pxQueue), SEGGER_SYSVIEW_ShrinkId((U32)pvBuffer))
//#define traceQUEUE_RECEIVE( pxQueue )
    SYSVIEW_RecordU32x4(apiID_OFFSET + apiID_XQUEUEGENERICRECEIVE,
    SEGGER_SYSVIEW_ShrinkId((U32)pxQueue), SEGGER_SYSVIEW_ShrinkId((U32)pvBuffer),
    xTicksToWait, xJustPeeking)
//#define traceQUEUE_RECEIVE_FAILED( pxQueue )
    SYSVIEW_RecordU32x4(apiID_OFFSET + apiID_XQUEUEGENERICRECEIVE,
    SEGGER_SYSVIEW_ShrinkId((U32)pxQueue), SEGGER_SYSVIEW_ShrinkId((U32)pvBuffer),
    xTicksToWait, xJustPeeking)
//#define traceQUEUE_RECEIVE_FROM_ISR( pxQueue )
    SEGGER_SYSVIEW_RecordU32x3(apiID_OFFSET + apiID_XQUEUERECEIVEFROMISR,
    SEGGER_SYSVIEW_ShrinkId((U32)pxQueue), SEGGER_SYSVIEW_ShrinkId((U32)pvBuffer), (U32)
    pxHigherPriorityTaskWoken)
//#define traceQUEUE_RECEIVE_FROM_ISR_FAILED( pxQueue )
    SEGGER_SYSVIEW_RecordU32x3(apiID_OFFSET + apiID_XQUEUERECEIVEFROMISR,
    SEGGER_SYSVIEW_ShrinkId((U32)pxQueue), SEGGER_SYSVIEW_ShrinkId((U32)pvBuffer), (U32)
    pxHigherPriorityTaskWoken)
#define traceQUEUE_REGISTRY_ADD( xQueue, pcQueueName )
    SEGGER_SYSVIEW_RecordU32x2(apiID_OFFSET + apiID_VQUEUEADDTOREGISTRY,
    SEGGER_SYSVIEW_ShrinkId((U32)xQueue), (U32)pcQueueName)
#if ( configUSE_QUEUE_SETS != 1 )
// #define traceQUEUE_SEND( pxQueue )
    SYSVIEW_RecordU32x4(apiID_OFFSET + apiID_XQUEUEGENERICSEND, SEGGER_SYSVIEW_ShrinkId((
    U32)pxQueue), (U32)pvItemToQueue, xTicksToWait, xCopyPosition)
#else
#define traceQUEUE_SEND( pxQueue )                                       SYSVIEW_RecordU32x4(
    apiID_OFFSET + apiID_XQUEUEGENERICSEND, SEGGER_SYSVIEW_ShrinkId((U32)pxQueue), 0, 0,
    xCopyPosition)
#endif
#define traceQUEUE_SEND_FAILED( pxQueue )                                SYSVIEW_RecordU32x4(
    apiID_OFFSET + apiID_XQUEUEGENERICSEND, SEGGER_SYSVIEW_ShrinkId((U32)pxQueue), (U32)
    pvItemToQueue, xTicksToWait, xCopyPosition)
//#define traceQUEUE_SEND_FROM_ISR( pxQueue )
    SEGGER_SYSVIEW_RecordU32x2(apiID_OFFSET + apiID_XQUEUEGENERICSENDFROMISR,
    SEGGER_SYSVIEW_ShrinkId((U32)pxQueue), (U32)pxHigherPriorityTaskWoken)
#define traceQUEUE_SEND_FROM_ISR_FAILED( pxQueue )
    SEGGER_SYSVIEW_RecordU32x2(apiID_OFFSET + apiID_XQUEUEGENERICSENDFROMISR,
    SEGGER_SYSVIEW_ShrinkId((U32)
```

This results in queue operations not being logged in the system viewer and less traffic for the system viewer.
Problem: component versino = 2.42, system view version = 2.52a

# 6 Hyperref

Ein mit der vorliegenden Vorlage erstelltes LaTeX-Dokument enthält zahlreiche Hyperref-Links. Dies bedeutet, dass sämtliche Verweise im PDF auch direkt als Link fungieren. Klickt man den Verweis an, landet man auf der entsprechenden Seite. Hyperref-Links funktionieren für:

- Bilder
- Tabellen
- Überschriften
- Gleichungen
- Codeblöcke
- Quellenverweise

## 6.1 Backref

In den Quellenverweisen ist jeweils die Informaiton enthalten, auf welchen Seiten auf diese Quelle verwiesen wird (Zitiert auf Seite ...). Auch diese Verweise funktionieren wiederum als Links im PDF.

## 6.2 Autoref

Bei Verweisen, welche mit der Funktion „\autoref“, bzw, „\aref“ oder „\autoeqref“ formuliert werden, wird beispielsweise der Begriff „Bild“ automatisch dazugeschrieben. Dabei fungiert zudem im PDF nicht nur die Zahl als Link, sondern auch der ganze dazugehörige Begriff (z.B. **??** anstatt nur **??**).das funktioniert natürlich auch mit:

**??**
**??**
Kapitel 6
Abschnitt 6.1
**??**

Im Fall von Anhängen wird der „\aref{ }“-Befehl (*a* für appendix oder Anhang) benötigt:

Anhang A
Anhang A.1

Bei Formeln kann der Befehl „\autoeqref“ angewendet werden:

Gleichung (**??**)

**Tipp**

- Labels mit einem Kürzel beginnen, das Auskunft darüber gibt, auf welche Art von Textbaustein es verweist (z.B. „picBeispiel“ für Bild, „tabBeispiel“ für Tabelle, „eqBeispiel“ für Gleichung, „refBeispiel“ für Überschriften).

# 7 Literaturverweise

## 7.1 Bibliography und Zotero

Die Einträge im Bibliography-File können mit Zotero erstellt werden. Wenn die entsprechende Literatur dort bereits eingetragen ist, kann sie einfach per Drag-and-Drop in das BibLaTex-Literaturfile gezogen werden. Als Alternative kann per Rechtsklick auf die Datei über den Befehl "ausgewählten Eintrag exportieren" ein neues BibLaTex-File mit dem Eintrag erstellt werden. Dies funktioniert auch, wenn mehrere Dateien angewählt sind.
Bei MSE-Berichten sind sämtliche Literaturstellen in der Zotero-Datenbank abzulegen. Zum Eintragen der benötigten Attribute (Titel, Autor etc.) kann Tabelle 7.1 konsultiert werden. Folgende sind Punkte zu beachten:

– **Bevor man bei Zotero eine Literaturstelle hinzufügt, ist zu prüfen, ob diese bereits existiert.** Allfällig bemerkte doppelte Einträge werden fusioniert.
– Der Name der heraufgeladenen PDF-Datei soll dem Schema „Jahr - Autor - Titel" folgen. Also zum Beispiel „2009 - Seelhofer - Ebener Spannungszustand im Betonbau.pdf". Bei MSE-Dokumenten schreibt man zusätzlich die das Modul dazu, also beispielsweise: „2013 - Stenz - VM2 - Kontinuierliche Spannungsfeldmodelle.pdf".
– Beim Eintrag einer Literaturstelle in Zotero ist unter „Datum" immer nur das Jahr einzutragen, Ausnahme: Zeitschriftenartikel (dort wenn vorhanden den Monat auch berücksichtigen).
– Bei Vertiefungsmodulen ist unter „Art des Berichtes" der Eintrag „Bericht Vertiefungsmodul 2" zu machen. Der Zusatz „Bericht" wird im Hinblick auf die Zitierung in LaTeX der Verständlichkeit halber benötigt.
– Beim Literaturtyp „Bericht" werden in Zotero „Seiten" (von-bis) und nicht die „Anzahl der Seiten" verlangt. Meistens soll im Literaturverweis aber „123 S." (Seitenanzahl) und nicht „S. 123-127" (gewisse Seiten eines Dokuments) stehen. Die erste Darstellung kann erzwungen werden, wenn in Zotero im Feld „Seiten" der Eintrag „123 S." und nicht nur „123" gemacht wird. Letzterer Eintrag würde zur meist unerwünschten Darstellung „S. 123" im Literaturverzeichnis führen.
– Um in LaTeX auf eine aus Zotero exportierte Literaturstelle zu verweisen, wird im Argument des \cite-Befehl folgendes Muster verlangt: „Autor"_ „1.Wort des Titels " _ „Jahr" . Beispiel: Auf „Ebener Spannungszustand im Betonbau" von Seelhofer (2009) wird mit „\cite{seelhofer_ ebener_ 2009}" zitiert.
– Achtung: In Zotero zusätzlich eingegebene Informationen (übrige, unbenutzte Felder) können unter Umständen auch in LaTeX im Literaturverzeichnis erscheinen (z.B. wenn bei einem Buch der ISBN eingegeben wird, wird dieser am Ende des Verweises im Literaturverzeichnis aufgeführt).
– Die Argumente „@keywords" und „@file" in BibLaTex-Literaturdatenbanken entstehen automatisch beim Export aus Zotero und haben keinen Einfluss auf den Output im Literaturverzeichnis. Sie können also in der Datenbank belassen werden.
– Bei Zeitschriftenartikeln muss bei Verweisen keine Seitenangabe gemacht werden, z.B. [10]. In allen anderen Fällen muss die Seitenzahl, von der die Information aus der Quelle entnommen wurde, angegeben werden, z.B. [11, S. 34] mit „\cite[S. 34]{seelhofer_ ebener_ 2009}"

| Literaturtyp | Typ Zotero | Typ LaTeX | Attribute | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | | Titel | Autor | Nr. Bericht | Art Bericht | Ort | Institution | Seiten | Anz. Seiten | Datum | Verlag | Name Konf. | Band | Ausgabe | Publikation |
| | | | title | author | number | type | location | institution | pages | pagetotal | year | publisher | eventtitle | volume | issue /number | journaltitle |
| Bericht [5] | Bericht | report | X | X | Nr. 75 | Bericht | X | X | 000 S. | | Jahr | | | | | |
| Buch [17] | Buch | book | X | X | | | | | | 000 | Jahr | X | | | | |
| Dissertation [11] | Dissertation | thesis | X | X | | | X | X | | 000 | Jahr | | | | | |
| Diskussionsbericht [6] | Bericht | report | X | X | Nr. 124 | Diskussionsbericht | X | X | 000 S. | | Jahr | | | | | |
| Konferenz-Paper, -bericht [14] | Konferenz-Paper | inproceedings | X | X | | | | | 00-00 | | Jahr | | X | X | | |
| MSE Master-Thesis [1] | Bericht | report | X | X | | Master-Thesis | X | X | 000 S. | | Jahr | | | | | |
| MSE Bericht VM1, VM2 [2] | Bericht | report | X | X | | Bericht Vertiefungsmodul 1 | X | X | 000 S. | | Jahr | | | | | |
| Norm [4] [8] [13] | Bericht | report | X | | | | X | X | 000 S. | | Jahr | | | | | |
| Norm Dokumentation [12] | Bericht | report | X | | | | X | X | 000 S. | | Jahr | | | | | |
| Anleitung / Manual [15] | Bericht | report | X | X | | Anleitung (o.ä.) | X | | 000 S. | | Jahr | | | | | |
| Versuchsbericht [3] [9] | Bericht | report | X | X | | Versuchsbericht | X | X | 000 S. | | Jahr | | | | | |
| Vorlesungsskript [7] | Manuskript | report | X | X | | Vorlesungsskript | X | X | 000 S. | | Jahr | | | | | |
| Zeitschriftenartikel [10] [16] | Zeitschriftenart. | article | X | X | | | | | 00-00 | | Monat.Jahr | | | V. 00 oder 00 | No. 00 oder 00 | X |

**Tabelle 7.1:** Für die Literaturverweise benötigte Informationen beim Heraufladen auf Zotero und Zitieren in LaTeX

# Anhang A Anhangstruktur

Hier sollte man am besten jegliche Teile über den \inlcude-Befehl importieren. Die Überschriften werden genau gleich wie beim Hauptteil des Berichts über die Befehle \chapter, \section, \subsection und \subsubsection eingefügt. Die Layoutstruktur ist analog zu den normalen Kapiteln:

## A.1 Unterkapitel im Anhang

Dies hier ist ein Blindtext zum Testen von Textausgaben. Wer diesen Text liest, ist selbst schuld. Der Text gibt lediglich den Grauwert der Schrift an. Ist das wirklich so? Ist es gleichgültig, ob ich schreibe: „Dies ist ein Blindtext“ oder „Huardest gefburn“? Kjift – mitnichten! Ein Blindtext bietet mir wichtige Informationen. An ihm messe ich die Lesbarkeit einer Schrift, ihre Anmutung, wie harmonisch die Figuren zueinander stehen und prüfe, wie breit oder schmal sie läuft. Ein Blindtext sollte möglichst viele verschiedene Buchstaben enthalten und in der Originalsprache gesetzt sein. Er muss keinen Sinn ergeben, sollte aber lesbar sein. Fremdsprachige Texte wie „Lorem ipsum“ dienen nicht dem eigentlichen Zweck, da sie eine falsche Anmutung vermitteln.

### A.1.1 Tieferes Kapitel

#### Noch tieferes Kapitel

Dies hier ist ein Blindtext zum Testen von Textausgaben. Wer diesen Text liest, ist selbst schuld. Der Text gibt lediglich den Grauwert der Schrift an. Ist das wirklich so? Ist es gleichgültig, ob ich schreibe: „Dies ist ein Blindtext“ oder „Huardest gefburn“? Kjift – mitnichten! Ein Blindtext bietet mir wichtige Informationen. An ihm messe ich die Lesbarkeit einer Schrift, ihre Anmutung, wie harmonisch die Figuren zueinander stehen und prüfe, wie breit oder schmal sie läuft. Ein Blindtext sollte möglichst viele verschiedene Buchstaben enthalten und in der Originalsprache gesetzt sein. Er muss keinen Sinn ergeben, sollte aber lesbar sein. Fremdsprachige Texte wie „Lorem ipsum“ dienen nicht dem eigentlichen Zweck, da sie eine falsche Anmutung vermitteln.

# Literaturverzeichnis

[1] Amsler, M., *Bemessung von Platten - Modelle und Beispiele*, Master-Thesis, Horw: Hochschule Luzern - Technik & Architektur, Kompetenzzentrum Konstruktiver Ingenieurbau, 2013, 105 S. (Zitiert auf S. 38).

[2] Amsler, M., *Verstärkung von bestehenden Betontragwerken mit Aufbeton*, VM1, Horw: Hochschule Luzern - Technik & Architektur, Kompetenzzentrum Konstruktiver Ingenieurbau, 2012, 74 S. (Zitiert auf S. 38).

[3] Amsler, M. und Thoma, K., *Durchstanzversuch mit Aufbeton - Versuch VA1*, Versuchsbericht, Horw: Hochschule Luzern - Technik & Architektur, Kompetenzzentrum Konstruktiver Ingenieurbau, 2013, 70 S. (Zitiert auf S. 38).

[4] *Eurocode 2: Bemessung und Konstruktion von Stahlbeton- und Spannbetontragwerken - Teil 1-1: Allgemeine Bemessungsregeln und Regeln für den Hochbau*, Lausanne: Europäisches Komitee für Normung, 2010, 246 S. (Zitiert auf S. 38).

[5] Grob, J., *Ermüdung von Stahlbeton- und Spannbetontragwerken*, Bericht Nr. 75, Zürich: IBK, 1977, 58 S. (Zitiert auf S. 38).

[6] Haller, P, *Schwinden und Kriechen von Mörtel und Beton*, Diskussionsbericht Nr. 124, Zürich: Eidgenössische Materialprüfungs- und Versuchsanstalt, 1940, S. 39, (Zitiert auf S. 38).

[7] Menn, C., *Langzeit-Vorgänge - Der Einfluss von Kriechen und Schwinden auf den Verformungs- und Spannungszustand von Stahl-Beton-Tragwerken*, Vorlesungsskript, Zürich: ETH Zürich, 1977, 69 S. (Zitiert auf S. 38).

[8] *Model Code 2010 - Final draft, Volume 1, fib Bulletin No. 65*, Lausanne: Fédération Internationale du Béton, 2010, 311 S. (Zitiert auf S. 38).

[9] Muttoni, A., Schwarz, J. und Thürlimann, B., *Bemessen und Konstruieren von Stahlbetontragwerken mit Spannungsfeldern*, Vorlesungsskript, Zürich: ETH Zürich, 1988, 122 S. (Zitiert auf S. 38).

[10] Rüsch, H., “Researches Toward a General Flexural Theory for Structural Concrete”, in: *Journal of the American Concrete Institure* 57 (No. 7 Juli 1960), S. 28, (Zitiert auf S. 37, 38).

[11] Seelhofer, H., “Ebener Spannungszustand im Betonbau Grundlagen und Anwendungen”, Diss., Zürich: ETH Zürich, 2009, 247 S., (Zitiert auf S. 37, 38).

[12] *SIA Dokumentation D 0192, Betonbau, Bemessungsbeispiele zur Norm SIA 262*, Zürich: Schweizerischer Ingenieur- und Architektenverein, 2004, 156 S. (Zitiert auf S. 38).

[13] *SIA Norm 262, Betonbau*, Zürich: Schweizerischer Ingenieur- und Architektenverein, 2013, 102 S. (Zitiert auf S. 38).

[14] Szépe, F., “Bemessung der Eisenbahnbrücken in Stahlbeton mit Rücksicht auf die Einschränkung der Rissbildung”, in: IABSE, Bd. Vol. 5, 1956, S. 843 –857, (Zitiert auf S. 38).

[15] Teschl, S., *Matlab - Eine Einführung*, Anleitung, Wien, 2001, 44 S. (Zitiert auf S. 38).

[16] Trost, H., “Auswirkungen des Superpositionsprinzips auf Kriech- und Relaxationsprobleme bei Beton und Spannbeton”, in: *Beton und Stahlbetonbau* 10 (1967), S. 230–238, 261–269, (Zitiert auf S. 38).

[17] Wehnert, M., *Ein Beitrag für drainierten und undrainierten Analyse in der Geotechnik*, Eigenverlag des Institus für Geotechnik, 2006, 167 S., (Zitiert auf S. 38).

# Bezeichnungen

## Lateinische Grossbuchstaben

| | |
|---|---|
| $A_c$ | Fläche eines Betonquerschnitts |
| $B$ | Belastungsgrad |
| $B_{cr}$ | Belastungsgrad bei Erreichen des Risslastniveaus |
| $E_c$ | Elastizitätsmodul von Beton |
| $M$ | Moment |
| P | Pol auf dem Mohrschen Kreis der Verzerrungen |
| $P$ | Einzellast |
| $\mathrm{P}_F$ | Pol auf dem Mohrschen Kreis der aufgebrachten Spannungen |
| $Q$ | Last, Belastung |
| $RH$ | Luftfeuchtigkeit |

## Lateinische Kleinbuchstaben

| | |
|---|---|
| $a_s$ | längenbezogener Bewehrungsquerschnitt |
| $c_u$, $c_o$ | Bewehrungsüberdeckung unten und oben |
| $c_{c\mathrm{I}ij}$ | Ungerissene Betonsteifigkeitsmatrix |
| $c_{c\mathrm{II}ij}$ | Gerissene Betonsteifigkeitsmatrix |
| $n_x$, $n_y$, $n_{xy}$ | Plattenschnittkräfte: Längenbezogene Normalkräfte |
| $q_x$, $q_y$, $q_z$ | Flächenlasten |
| $s$ | Beiwert Abbindegeschwindigkeit |
| $s_{rm}$ | diagonaler Rissabstand |
| $t_s$ | Zeitpunkt des Schwindbeginns |
| $u$ | Umfang des Betonquerschnitts |
| $x$, $y$, $z$ | Kartesische Koordinaten |

## Griechische Grossbuchstaben

| | |
|---|---|
| $\Delta\sigma_{ci}$ | Tensor Änderung der Betonspannungen |

## Griechische Kleinbuchstaben

| | |
|---|---|
| $\alpha$ | Faktor Abbindegeschwindigkeit, Drehwinkel Transformation |

| | |
|---|---|
| $\varepsilon_{cs}$, $\varepsilon_{csi}$ | Schwinddehnung bzw. Schwinddehnungstensor des Betons |
| $\varepsilon_{cs,\infty}$ | Endschwindmass |
| $\rho_x$, $\rho_y$ | geometrischer Bewehrungsgehalt in $x$-Richtung bzw. in $y$-Richtung |
| $\varphi$ | Kriechzahl |

## Sonderzeichen

| | |
|---|---|
| $\varnothing_x$, $\varnothing_y$ | Stabdurchmesser der Bewehrung in $x$-Richtung bzw. in $y$-Richtung |
| $\partial$ | Differenz bei der partiellen Ableitung |
| $\infty$ | unendlich |

## Abkürzungen

| | |
|---|---|
| CMM | Gerissenes Scheibenmodell |
| Emat | Steifigkeitsmatrix (Jakobimatrix) |
| GH | Modell für gerissene Hauptrichtungen |
| LE | Modell für linearelastisches Verhalten |
| MC | Model Code |

# Lebenslauf

## Personalien

| | |
|---|---|
| Name | Peter Muster |
| Adresse | Bahnhofstrasse 1<br>6004 Luzern |
| Geburtsdatum | 01.01.1989 |
| Heimatort | 6004 Luzern |
| Zivilstand | ledig |

## Ausbildung

| | |
|---|---|
| August 1996 - Juli 2005 | Primar- und Sekundarschule, Dallenwil |
| August 2005 - Juli 2009 | Lehre als Bauzeichner mit technischer Berufsmaturität<br>Biegebruch GmbH, Luzern |
| September 2009 - Juli 2012 | Bauingenieurstudium Bachelor of Science<br>Hochschule Luzern - Technik & Architektur, Horw |
| September 2013 - Februar 2016 | Bauingenieurstudium Master of Science<br>Vertiefung im Konstruktiven Ingenieurbau<br>Hochschule Luzern - Technik & Architektur, Horw |

## Berufliche Tätigkeit

| | |
|---|---|
| Juli 2010 - August 2010 | Bauzeichner bei Schubversagen AG, Luzern |
| Juli 2011 - August 2011 | Hilfsassistent Abteilung Bautechnik,<br>Hochschule Luzern - Technik & Architektur, Horw |
| Dezember 2012 - September 2015 | Assistent Abteilung Bautechnik,<br>Hochschule Luzern - Technik & Architektur, Horw |

# Liste der noch zu erledigenden Punkte