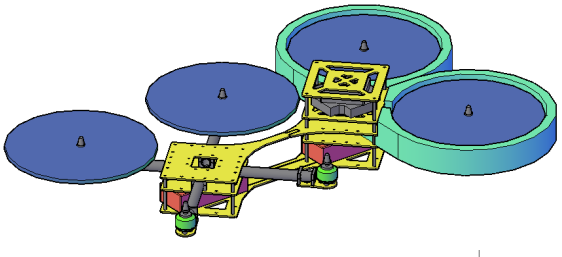
Hochschule Ulm

Project BumbleBee

Department of Computer Science

C.Löser
C.Müller
D.Klitzke
H.Heinisch
S.Krais
Y.Groner

1st Advisor: Prof. Dr. Strahnen
2nd Advisor: Prof. Dr. Steiper



Duration: 01. March 2014 – 30. January 2015

Contents

1. Introduction

In todays modern high-tech world there are more and more fields of application for so called multicopters. These are aircrafts that are driven by multiple rotors, are relatively easy to control and are able to hover on one spot which even allows them to navigate through narrow and inaccessible areas. Typical applications of multicopters can be found in several areas like photography, emergency management and even parcel delivery. Thereby the aircraft has not to be necessarily controlled by a human with help of a remote control. Instead there are also approaches in which the aircraft navigates through the area autonomously.
Outdoor the aircraft navigation is relatively easy because the aircraft can locate itself easily via GPS. The indoor navigation becomes far more complicated because GPS doesn't work indoor since the system has to have visual contact to at least 3 satellites. Thus other navigation techniques like visual navigation over cameras and depth of field sensors have to be used which are intensively researched at the moment.
These alternative navigation techniques require very high processing performance. A common approach is to split the necessary computations on multiple independent processor cores. This leads to the problem that the different cores should access separate memories in order to avoid access contentions. These conflicts would reduce the performance and complicate the predictability within a real-time system. Therefore the memories of the the different processor cores should be separated or alternatively each processor has to have a well dimensioned cache.
In order to allow research in this field, a system should be built up preferably modular so that various sensors and actors can be installed for testing. A system for such research purposes should be built up in this project. Detailed information regarding the project goal can be found in the next section.

2. Project Description

2.1 Project Goal

n the project a control system for an autonomous multicopter shall be realized. It should provide functionality for flight stabilization, collision detection and additional computations like map creation and localization. In order to provide enough processing power it should consist of multiple processors that access separated memories in order to prevent any access conflicts. These could slow down or block the flight control system which is operating under real-time requirements. To allow tests of the control system a physical model shall be constructed that carries all hardware components.
In addition the system should be also usable as a development and research system which requires a high degree of modularity and extensibility. This point has to be realized in different aspects for example the physical model and the hardware of the control system. The physical model should e.g. allow the installation of additional sensors or additional accumulators. The control systems hardware interfaces have to be extendable to allow the connection of additionally placed sensors. An additional requirement of the customer is that the communication between different processor cores should be realized through a standard interface. A solution for this is provided by the use of the "Multicore Communications API" (MCAPI) which is manufactured by "The Multicore Association" and was already used in former projects of our customer. To make use of this knowledge and gather additional experience the use of this specific software is already defined in the project requirements.

2.2 Project Members

The Project was realized by the following project members.

Yvette Groner

In the project she mostly worked on the circuit board especially the power supply circuit for the multicopter.
Also the soldering of most circuits was done by her.
She also did most of the email communication with the customer.

Hannes Heinisch

In the project he mostly worked on the circuit board layout. He also has done the design of some interface circuits and also programming work regarding the drivers for some sensors. He also has made some changes at the SoPC.

Daniel Klitzke

In the project he mostly worked on the design of the SoPC especially the separation of memories and integration of IP-Cores. He also did some software design and programming work.
Also the coordination in meetings with the customer was done by him.

Simon Krais

In the project he mostly worked on the selection of parts and the construction of the physical model.
He also did the testing of several parts and was also responsible for the design and realization of the report via LaTeX.

Christoph Loeser

In the project he also mostly worked on the selection of parts and the constuction of the physical model which he also planned in a CAD program.
He also worked on the programming of several drivers.

Christian Mueller

In the project he mostly worked on the linux driver needed by the MCAPI.
He also did work on the SoPC and tested the MCAPI communication with different SoPC versions.

3. Requirements

3.1 Analysis of Customers Needs

The system has to carry a payload of minimum 1 kg. Therefore normally 6 to 8 rotors are needed and it has to be evaluated which number of rotors fits our requirement best by measuring the lifting capacity of selected motors and rotors.
Another requirement of the customer is that the model should fit through a standard door. Because of this the model has to be constructed with a maximum width of 85cm to have enough clearance.
The system also has to reach a flight time of 10 to 20 minutes. For multi-copters normally Lithium Polymer accumulators with 1 to 10 cells and 500 to 20000 mAh are used and it has to be measured how much power is consumed by the system, especially the motors. To reach these requirements the weight of the model should be as lightweight as possible. All components have to be checked regarding to their weight and the use of different materials such as carbon fiber should be evaluated.
The Customer also wants to have a modular design of the whole system.
Therefore a physical model has to be designed in order to have enough space for additional modules such as new sensors. It also has to be possible to change the weight distribution to keep the model balanced and also the electronics need to have enough standard interfaces to add new hardware components.
Another requirement of the customer is that the different software components don't interfere with each other. To fulfill this, the system should consist of multiple processors that have separated memory and interact with each other over bridges.
The customer also wants to have the possibility to extend the existing multiprocessor system with more powerful hardware over a widely spread communication protocol.
To meet this requirement an Ethernet interface should be realized and the system should support to give the new hardware access to the required sensors.
To meet security requirements manual interaction has to be possible at all times.
Therefore the system has to have a receiver for a remote control and has to meet hard real-time requirements.
The system also should be able to fly stable and to give the other processors the possibility to control the flight of the system. Therefore a flight control unit has to be designed that has an interface with which other processors can interact.

3.2 Functional and Non-funtcional Requirements

Functional Requirements

- **The flight control of the model consists of a multiple processor system**
  Description: The client wants a multi processor system in order to have separate address spaces for different processes.
  Rationale: The flight control system consists of different processors.
  Priority: High

- **Option to replace the main processor**
  Description: It would be nice, if it will be possible to replace the actual main processor with a more powerful one.
  Rationale: It is possible to use an external processor system for more computing power.
  Priority: Mid

- **The model is motor-vibration resistant as good as possible.**
  Description: The clients mapping algorithm needs a preferably stable flight for calculating a map. High vibrations through the motors or propellers should be avoid.
  Rationale: Pulse dampers are integrated in the model.
  Priority: Mid

- **The model has an attachment for mounting Kinect cameras.**
  Description: The clients application for the model needs cameras, actually connect cameras, for creating 3D maps from the environment (in buildings). Therefore it should be possible to mount these cameras in a proper way on the model.
  Rationale: The model has an attachment for cameras.
  Priority: Mid

- **The model should be able to communicate later on with a basis station.**
  Description: The ability for using different communication standards for data transmission should be considered
  Rationale Interfaces for using communication techniques are implemented.
  Priority: Mid

Non-functional Requirements

- **The model should be usable for indoor flight.**
  Description: The client wants to use the model for indoor navigation and mapping. Therefore the design needs to be as slim as possible. The model should be able to pass doors and smaller parts of a room.
  Rationale: The width of the design is lower than 80cm. The model is able to do a precise flight.
  Priority: Indispensable

- **The model should have an adequate flight time.**
  Description: The client needs an adequate flight time for his mapping-application. To be useful the model should be able to map bigger areas before it has to fly back for loading the power supply.
  Rationale: The flighttime is higher than 15 minutes.
  Priority: High

- **The model consists of a modular design.**
  Description: Actually the clients mapping algorithm works with two Kinect cameras. So the system has to carry these and a small computer for the mapping algorithm.

Later on it would be nice to have the possibility to replace single components and add additional sensors.
Rationale: The finished model can easily be modified and reshaped. Additional sensor mount points are available.
Priority: Mid

- **The model should be able to carry at least 1 kg of payload.**
Description: The client wants to carry some additional parts like the cameras and a small computer for map calculating. Therefore the model should be able to carry this additional payload easily.
Rationale: The finished model carry at least 1 kg payload
Priority: Indispensable

- **It is possible intervene the autonomous flight every time**
Description: It is necessary that the client is able to intervene the manual flight every time he wants. Therefore the difference between manual and algorithmic input has to be detected. Manual input must be accepted at all times with an higher priority.
Rationale: A reliable way to manually interact with the model has been implemented.
Priority: Indispensable

- **The model should be able to fly autonomously**
Description: The mapping of a building does mostly not take place in the clients view. So the model should be able to fly autonomous, controlled by the mapping algorithm.
Rationale: An interface for an autonomous flight is implemented.
Priority: High

- **The project has a good documentation for further project groups and an easy use for the client.**
Description: The project is not finalized in this projectteam, so it is necessary that the progress is good documented for further work on the project.
Rationale: HowTos and a detailed report are provided by the project team.
Priority: High

- **The clients are satisfied by the work of the project team in WS 14/15**
Description: The clients want to have a usable platform later on. Therefore it is necessary to create a proper design for the platform. Furthermore the project should be well documented.
Rationale: Clients are happy about the result.
Priority: Indispensable

4. System Architecture

4.1 System Overview

In this section an overview of the system is given. The system consists of two boards,which are needed to connect various hardware devices and can be mounted on a physical model. For more genereal information also have a look on[Bra06][HLN12][Wie12][BST10] and on [Hau13]. In illustration 4.1 you can see our ideo for a modular structure.

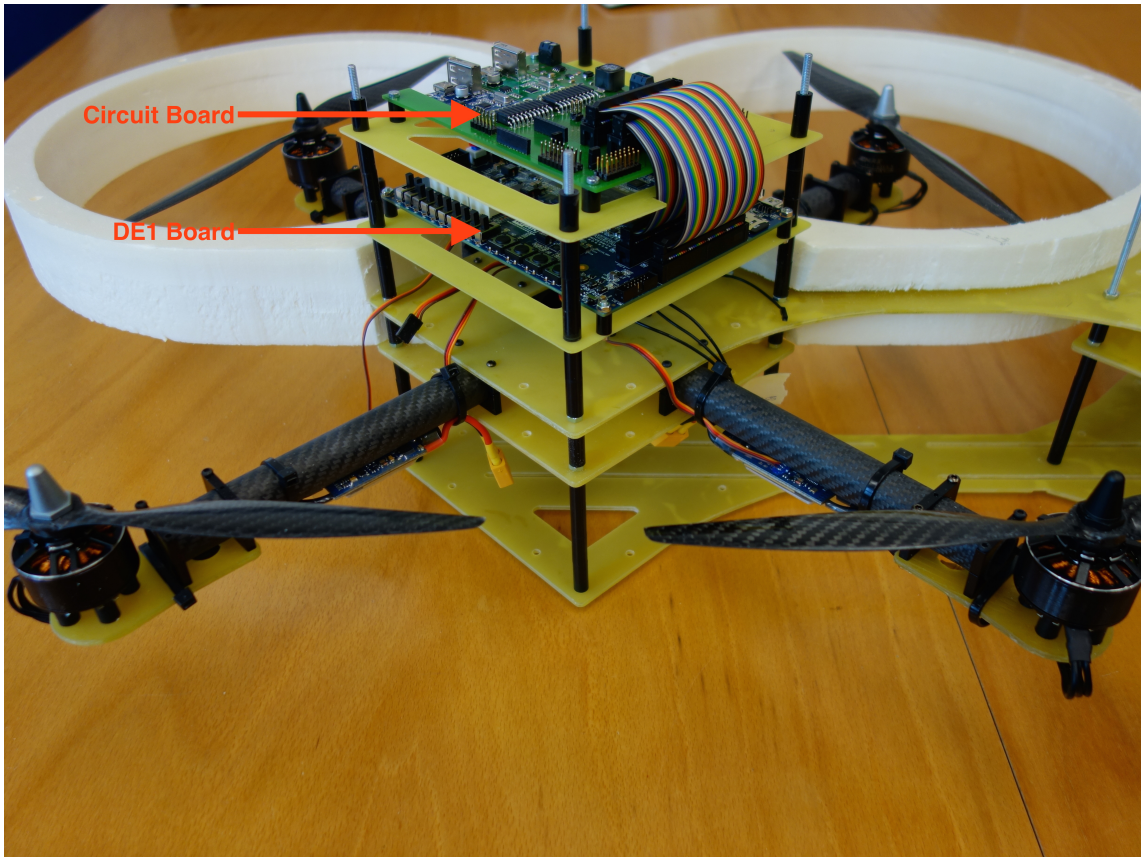


Figure 4.1: photo of one of the stacks on the model

The main board which hosts all three computer systems is the DE1-SoC board which,

for this purpose, hosts an Altera FPGA. The circuit board is connected to the DE1-SoC board and provides different interface circuits. Those are mostly needed for connecting the necessary hardware devices like sensors and actors and providing a stable power supply.
The DE1-SoC board can be divided into three different computer systems. Those are connected through FIFO bridges in order to provide the possibility of buffered comunication between them. The first system is the flight control. It provides the functionality to keep the aircraft stable in the air.
The second system is the collision detection which has the functionality to detect objects near the aircraft in realtime to avoid collisions with the environment.
The third system is the computation system which provides the functionality to create maps of the environment and to locate the aircraft in these maps.
Both boards can be mounted on the modular physical model, which consists of a frame that is able to carry all necessary sensors and actors. The physical model can be used as a multicopter with four or six rotors.

4.2 Used Hardware

4.2.1 Mainboard

The main board that is used in the project is a DE1-SoC board manufactured by Terasic Technologies Inc.. It is offered through the Altera University program which offers the possibility to obtain it very cheapely. In illustration 4.2 you can Alteras DE1-SoC Board.

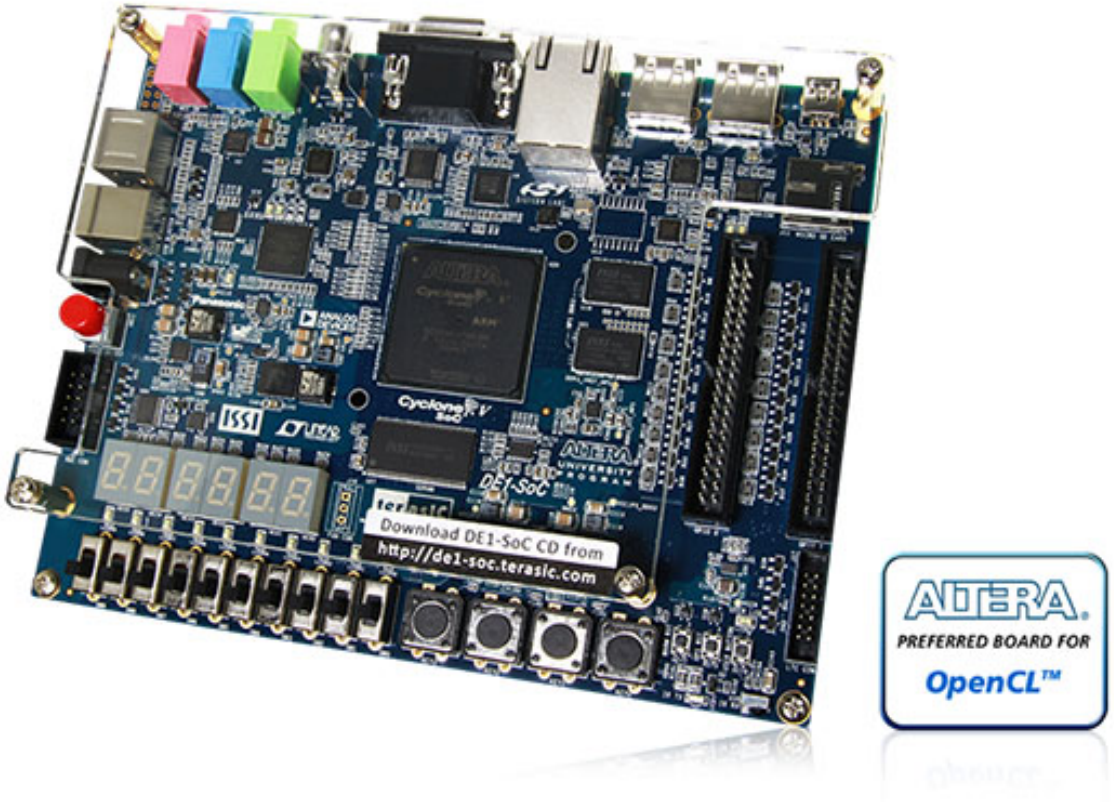


Figure 4.2: Alteras DE1-SoC Board

As its main components it contains an Altera Cyclone V SE 5CSEMA5F31C6N device. This is a FPGA manufactured by Altera that, additionally to the free programmable logic, contains a hard processor system containing a Dual-core ARM Cortex-A9 processor. Following the features of these two parts of the system are described more detailed.

Also important hardware components that are connected to one of these two system parts are listed below.[DE115]

4.2.1.1 FPGA

- 85000 logic elements
- 288 IOs
- 4450 kbits memory elements
- Multiple connected hardware devices
  - USB Blaster II for JTAG programming
  - 64 MBytes external SDRAM (16-bit data bus)
  - 2 hard memory controllers
  - Four 50MHz Clock Sources
  - Two 40 Pin expansion headers
  - Various Inputs such as switches and push buttons

4.2.1.2 HPS

- 800MHz Dual-core ARM Cortex A9 processor
- 1 Gbyte DDR3 SDRAM (32-bit data bus)
- Multiple connected hardware devices
  - 1 Gbit ethernet controller
  - 2-port USB host
  - Micro SD card socket
  - UART to USB

For a complete listing of all hosted hardware components or a detailed description of each component see [DE115].

In the project the FPGA part of the system is mostly used to build up two NIOS-II soft processors that run the flight control and the collision detection, while the HPS system is reserved for computational work that requires high performance.

4.2.1.3 Further Information Sources

- More detailed component descriptions can be found in [DE115]
- For the description of the built SoPC see 4.3
- For a tutorial how to program the board and continue work on the SoPC seeHowTo

4.2.2 Extension Board

A lot of different interfaces like two USB hosts, Micro SD Card, Ethernet, I2C or SPI are already implemented on the main board, the DE1-SoC board. For these controllers which are integrated into the FPGA additional interface circuits are necessary. Therefore it was necessary to create an extension board which provides different interface circuits and a power supply system.

To gain a high flexibility of the system some standard interfaces were implemented. I2C, SPI and UART are set up on the board. With this interfaces it is possible to connect various sensors to the system.

For the ability to use different wireless standards a XBee module is integrated on the board. Digi International offers various Xbee modules for different wireless connections. Starting with a module for Peer-to-Peer communication and ending with a module for the ZigBee standard.

The DE1-SoC board actually provides two USB ports for free use, but two other USB ports have to be implemented on the extension board. So additional potential to add different devices like a memory stick is given.

To ensure a save flight it has to be possible to overwrite the autonomous systems actions with a remote controller. Therefore an interface to connect an eight channel RC receiver is integrated in the extension board. Also a channel header is provided for maximum of eight motors.

The battery provides a voltage value between 19 and 26 volt, but the DE1-SoC board expect a value of 12V. Furthermore the board consumes 3.5A. Therefore it is necessary to have a voltage transformer. All extra devices, which can be added to the extension board can be supplied with 3.3V, 5V or 12V.

Finally all remaining free pins of the DE1-SoC board headers are wired to a header on the extension board.

In illustration 4.3 all interfaces of the DE1-SoC board and the extension board are shown.

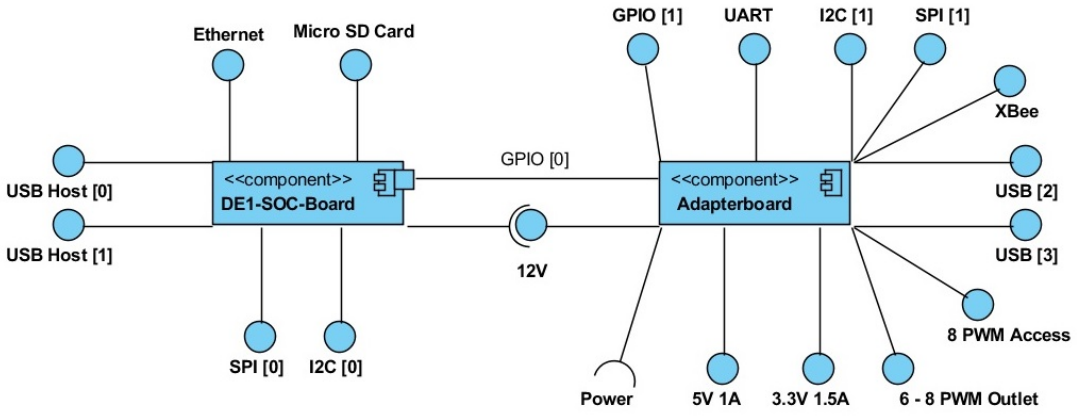


Figure 4.3: Interface definition

Most of the interfaces will be realized as IP-Cores in the FPGA. This reduces the size and

Interface	Number of Pins
I2C	2
SPI	4
UART	4
USB controller	12
XBee	4
PWM Input	8
PWM Output	8

Table 4.1: Overview of interfaces

weight of the extension board and dds some extensibility to the system. All signal pins on headers can be redefined in the FPGA if some interfaces have to be modified or changed.

A full listing of the count of the pins needed for the interfaces is shown in table 4.1.

I2C

For the I2C interface a header exists on the extension board. It provides the signal pins SDA and SCL from the FPGA. The I2C interface is used for the 9DoF Stick.

SPI

SPI is a widely used interface in embedded systems. The extension board provides a header to connect some SPI devices. For a SPI connection the four signals MOSI, MISO, SS and CLK are relevant. They are arranged on the header like shown in illustration 4.4.

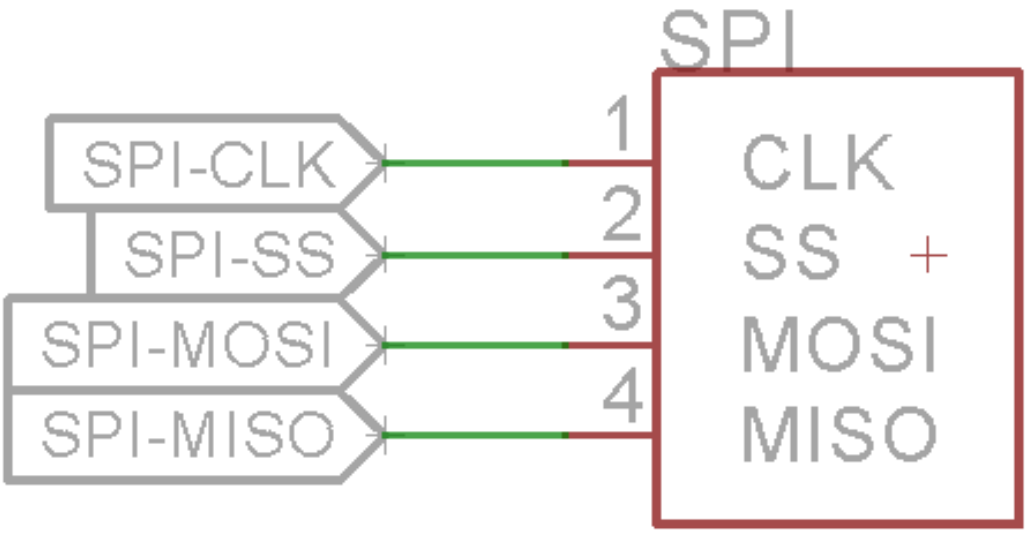


Figure 4.4: SPI pin schematic

UART

The extension board has one UART interface mapped to a header. The interface is connected to the FPGA the extension header of the DE1-SoC board. The interface supports

hardware flow control. For the UART interface a six pin header provides the signals RX, TX, RTS, CTS and also the two pins for the power supply, 5V and GND. Illustration 4.5 shows the pin shematic for the UART interface.

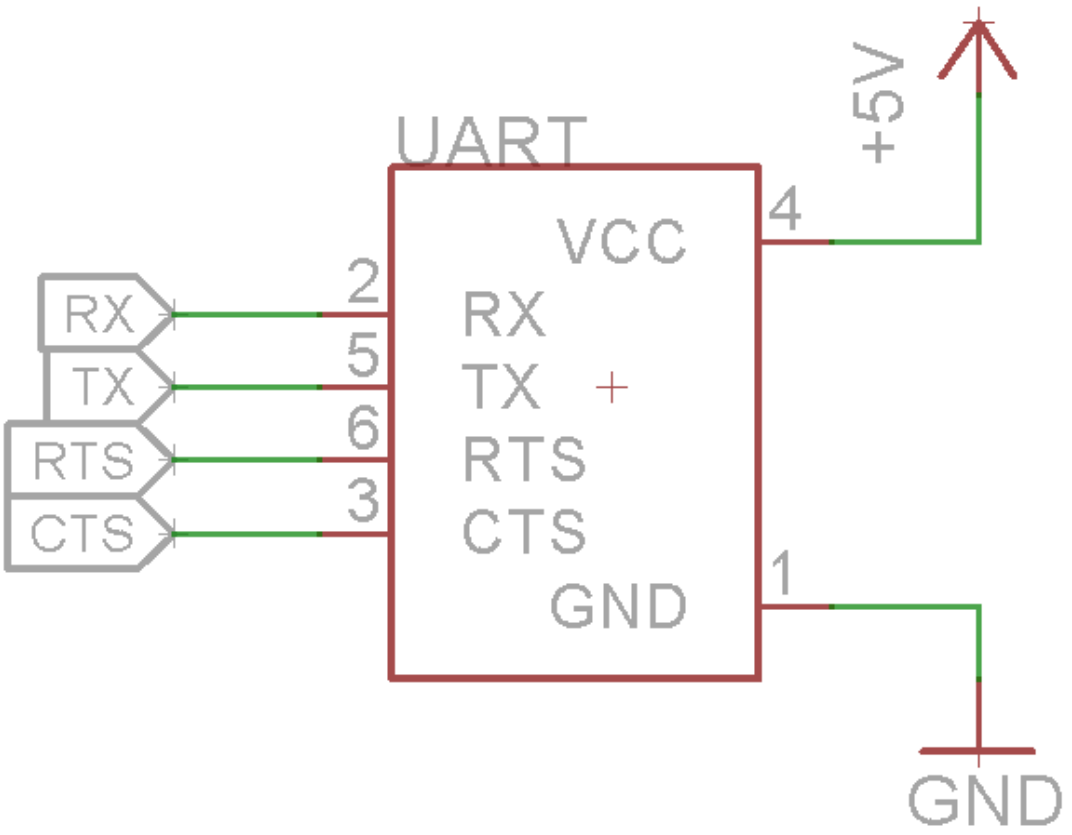


Figure 4.5: UART pin schematic

USB

For the USB interface it was impossible to use the same composition that Frank Seifert used in his bachelor thesis, because the chip ISP1362 he suggested is not produced anymore. One alternative is to use the USB 3300, the same one used on the DE1-SoC board. The advantages are that a Linux driver exists and it would be consistent. A circuit diagram is drawn in Eagle, but the components have to be dimensioned correctly. The disadvantage is that it was impossible for us to solder the microchip on our own. Therefore we decided to buy two extension circuit boards which use the USB 3300. Each board with one USB port an adequate circuit.

The extension board applies an ULPI interface. This interface is a low pin UMTI interface. Unfortunately no IP-Core for the ULPI interface exists for free. Altera offers a license for 5 000$. For more information see [USB15] and [OFF15].

An alternative solution is to transform the ULPI to UMTI via a wrapper. This isn't either tested or further tracked.

Xbee

To provide a variety of different wireless standards a XBee socket is integrated on the extension board. Thereby it is possible to connect different XBee modules to the FPGA.

A number of important pins is wired directly to the FPGA. The remaining pins are mapped to a 12 pin header. If some of them are needed later they could be connected in the SoPC. More information and a list of XBee modules is shown on the official site [XBE15]. The connection diagram in illustration 4.6 shows the complete wiring off the extension board.

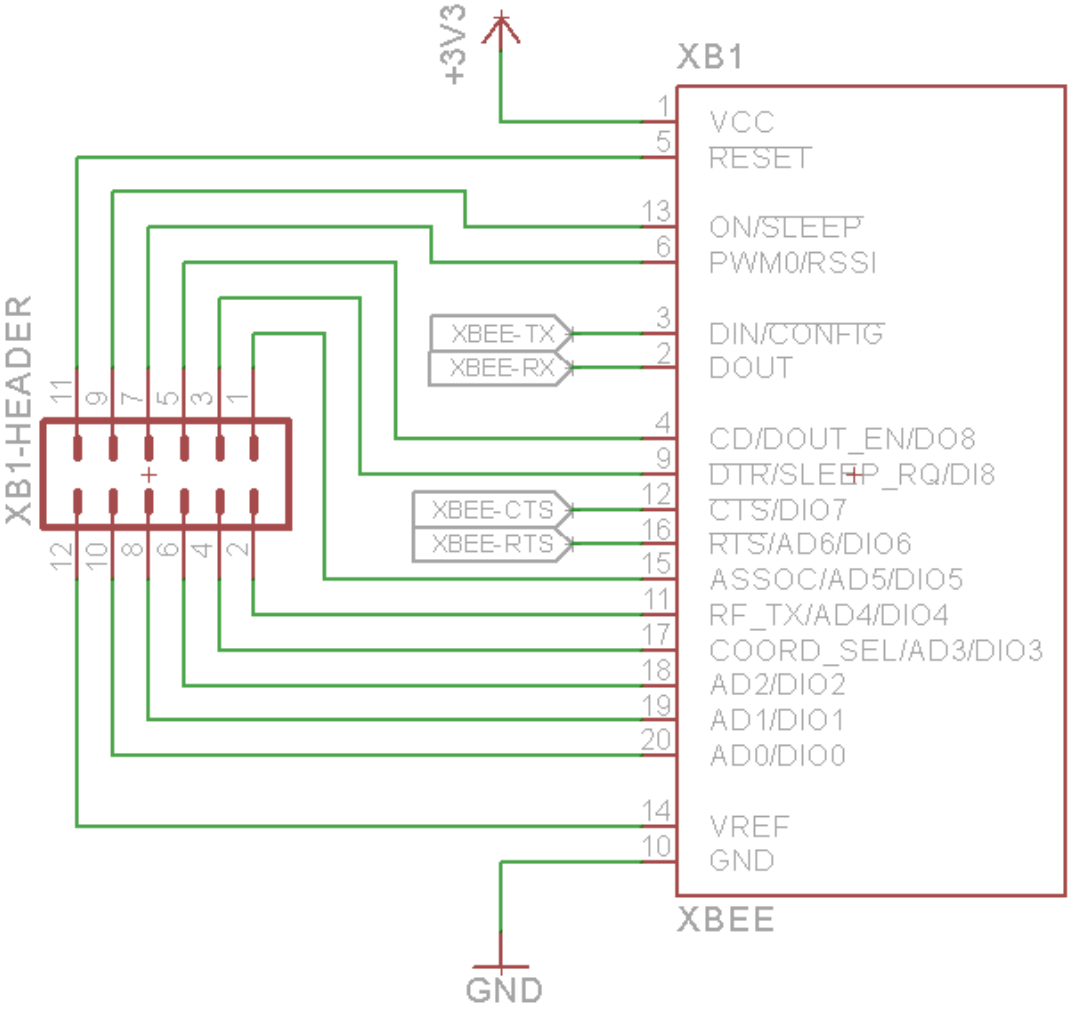


Figure 4.6: Xbee module circuit plan

PWM Input/Output

It should be possible to control the multi-copter with a RC remote controller. Therefore one 8x3 pin header is set up on the board. It provides 3 rows with 8 pins each. One row for the signals and the other two for 5V and GND. The voltage line can be switched off with a jumper on the board. Illustration 4.7 shows the pin assignment for the PWM interface.

Another header with the same dimensions provides pins to connect up to 8 motors to the model. If required it's also possible to switch off the power on the second line.

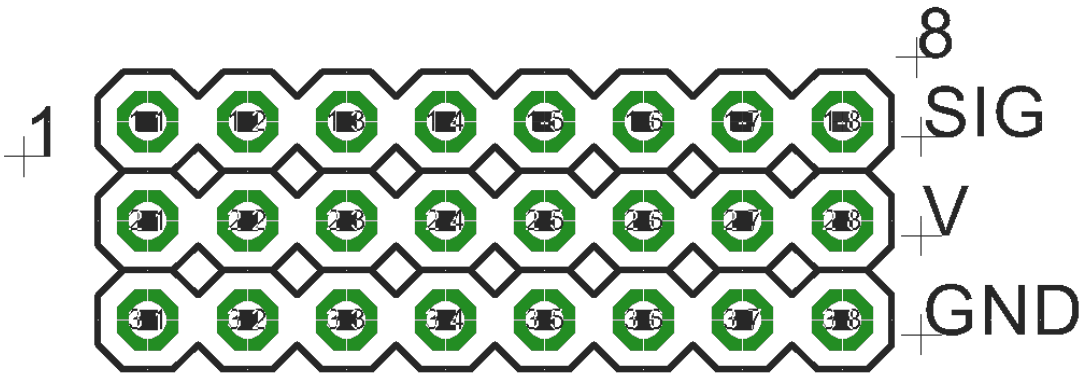


Figure 4.7: PWM pin assignment

Extension Header

The board provides a 30 pin header for all free connection pins. On the DE1-SoC board there are two 40 pin headers with 36 user pins. For all interfaces 42 pins are needed, so 30 pins are left for free use on the extension board.

Power Supply System

The power supply system is needed to pull-down the input value for the DE1-SoC board.

For more information hhave a look on 4.6.2.

4.2.3 Sensors

For a flight control system many different sensors are required to help stabilizing the model. Many producers already offer additional boards that mostly provide three sensors. We have chosen the 9 Degree of Freedom (9DoF) Stick from SparkFun. It includes an ADXL345 accelerometer, a HMC5883L magnetometer and an ITG-3200 MEMS gyroscope. They can be accessed over an I2C interface which is very simple to connect to our main board.

A second reason for this stick is that it was already used in several projects at the University. So different code samples already existed. Also a program for the Arduino board to test the functionality already exists. This helps when testing our code, because we can compare our measurement values with those the Arduino provides.

The stick provides a mounting hole to ensure proper installation and four pins for the wiring (VCC 3.3V, GND, SCL and SDA).

4.2.3.1 Accelerometer

With this sensor we are able to measure proper acceleration. It is a 3-axis accelerometer that can measure up to +-16g of force. If put at rest, it measures an upwards acceleration. This is due to earth gravity. The output data is formatted as a 16-bit twos complement number and is provided through 2 separate 8-bit registers for each axis.

4.2.3.2 Gyroscope

The gyroscope helps to detect rotational movement for all three axis. The chip includes three analog to digital converters. To avoid the need for an extra calibration, the chip has a high temperature stability. A temperature sensor is also provided if calibration should become necessary. If needed, a digitally programmable low-pass filter is also available.

4.2.3.3 Magnetometer

The magnetometer is used to measure the magnitude and direction of the Earth's magnetic field to provide us with reliable data regarding our direction in three axis. The accuracy ranges from two milli-gauss to 8 gauss. The measurements have a 1 degree to 2 degree Degree accuracy and are provided by a 12-bit analog to digital converter.

4.2.4 Actors

4.2.4.1 Motor

Motors for modelbuilding

Brushless motors are the most common motordesign in use for modelbuilding (For more information visit [Ayd15][Bue]). The brushless design differs from a "'normal"'brushed motor in some points. The three phases of current are connected with the coils from the outside. This motor type can not be connected directly to (direct) current, you need a regulator which generates a tri-phase alternating field. This regulator is placed in the actuator. The actuator is connected with the motor over three wires. Normally the actuator is a part located in the ESC which is described further down. Modern brushless motors for modelbuilding work sensor-less. For detecting the position of the stator they use the electromagnetic induction in the stator-coils (generator principle). If you turn the shaft, the rotation resistance will raise up when the propeller-magnets pass the stator-coils. Some alternating voltage will be induced which can be measured by the connected actuator. The mechanical composition is divided in two different options:

- In-runner:
  The coils are placed at the outside (in the motor-case). The permanent-magnets are fixed on the shaft, which rotates in an alternating field. This alternating field is generated by the coils. In-runners work very efficient. Furthermore they are smaller, turning faster and have a lower moment of force than out-runners. Because of that, the in-runner design is not usable for our model.

- Out-runner:
  The permanent-magnets are placed at the rotatable motor-bell. The motor-bell (propeller) itself is fixed at the shaft and rotates around the coils. The coils are fixed inside at the stator. The Out-runner design is easy to produce and cheap. Nearly all motortypes for multi-copters are designed as out-runners. In illustration 4.8 you can see the structure of an outrunning motor.

Each BL-motor is defined by some data

- KV - rotations per minute with 1 V
- idling cycle current - current without load (ideally 0 A)
- inner resistance - defined the inner waste of power
- maximal constant current - is influenced by thermal design, and waste of power
- moment of force - maximal constant power/KV

Demands to comply

The UAV of this project has to comply a lot of requirements. Many of them influence the choice of the right motor and propeller:

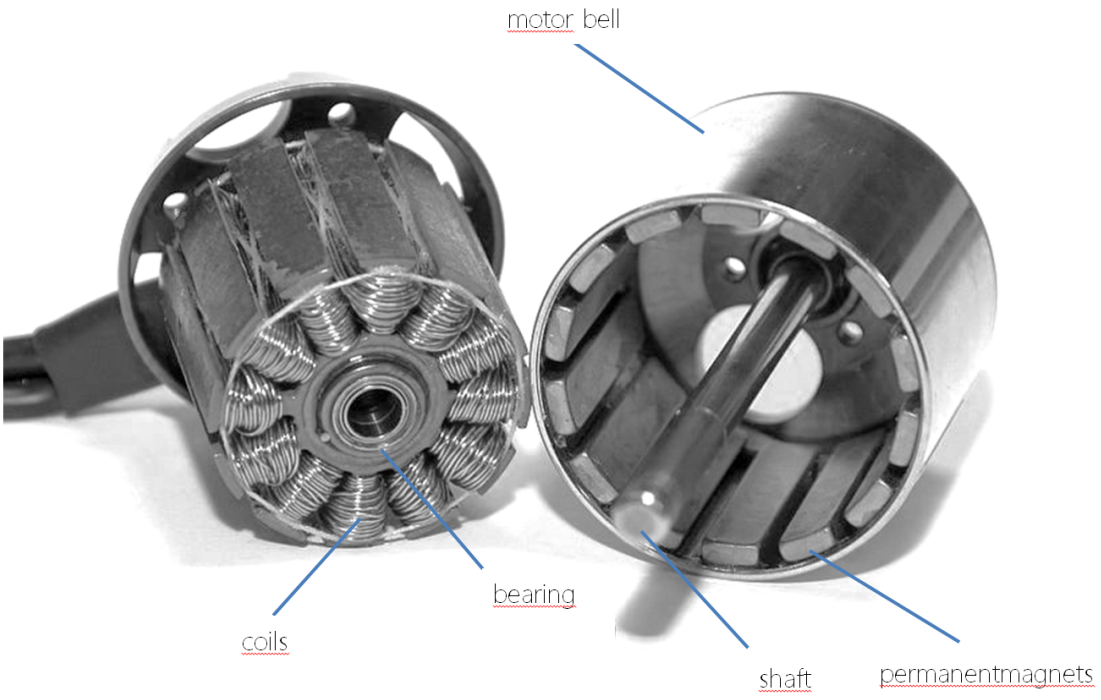


Figure 4.8: Schematic of an outrunner BL-motor

- Basically the boost of the drive system is a product of the propellers dimensions and the turnrate of the motor. The UAV should be indoor usable, so the dimension of the propellers is limited (by the width of a standard door). Assuming a normal door has a width of 90 cm and we need a tolerance of about 10 cm (because of some inaccuracy of the flight control), we have a maximal width of

  $B_{max} = \frac{80cm}{2.54cm} = 31.496inch$

  The model works with two parallel propellers

  $B_{propeller} = \frac{29.52inch}{2} = 15.748inch$

  For the bumper and a little spacing we defined 1.5 cm for each propeller side. So the upper limit for the propeller width is

  $B_{propeller} = 15.748inch - \frac{4*1.5cm}{2.54} = 15.748inch - 2.362inch = 13.386inch$

  The consequence is that the usable propellers for the UAV can have a maximum width of 13inch.

- The generated boost should be at least twice as high as the weight of the model is. Therefore is an advance estimatation of the weight necessary (table 4.2).

- Due to the definition of the used voltage system in section4.6.1 the motors should work as good as possible with a voltage level of 22.2V. Therefore the motor should have a proper KV value. Problems with the KV value can be:
  - If the KV value is too high, the motor requires smaller propellers. Normally smaller propellers with a higher turnrate are working less efficient than bigger

Component	weight [g]	amount	sum [g]
motor	140	6	840
propeller	20	6	120
ESC	30	6	180
extension board	100	1	100
SoC board	200	1	200
power distribution	100	1	100
sensors	10	4	40
small parts	300	1	300
frame	1000	1	1100
bumper	40	6	240
camera holder	450	1	450
i7 board	350	1	350
camera	433	2	866
sum			4886

Table 4.2: Calculation of the models weight (without payload)

ones with a lower turnrate. Furthermore this will result in a oversensitive flight behaviour due to some inaccuracy of the ESC.

  - If the KV values is too low, the motor requires bigger propellers. However the size for a propeller is limited by the possibility to pass normal doors. Furthermore a low KV value results in a lazy flight behaviour.

- The motor should have a proper size and the ability to do self-cooling with the propeller. Furthermore there should be a simple possibility to fix the motor onto the model.
- The motor should be specified for different operating grades by the manufacturer. Although these values are normally simplified they can provide a first evaluation. If there is no specification available for a voltage level this does not unconditionally mean that the motor will not work with this level but it is not tested with it and thus there can be some strange behaviour.

Proper models

The market of modelbuilding has become very big in the last years. There are a lot of manufacturers and even more sellers who sometimes sell the same model under different names. So it is necessary to border our focus on different models (table 4.3).

The manufacturers specify their models with some parameters, for example the current, the power, the thrust and the relation between power and weight for some throttles. As already said these are best case parameters which will not be reached in a real environment. In Table 4.3 you can see the motor models we have chosen in a first opening.

The T-motor U3[U315] is a very robust motor which is also usable outdoor. This model has a good resistance against water, dust and steam. But there are no values for a voltage supply system higher than 4S mentioned. Also the seller we have called could not provide any information about the behaviour with a higher supply voltage. In efficiency (g/W) in the 4S class this motor is the best one. However, this motor is very expensive and because of the missing specification we decided that we will not use this motor.

The BE4108[BE415] is a model which is proposed by our client. We think this model

14.8V	**T-Motors U3**	**BE4108**	**Emax MT 3515**	**T-Motors MT3515**
configuration	14.8V/13*4.4	14.8V/13*6.5	14.8V/13*4	14.8V/13*4.4
Thrust [g]	1800	1225	1650	1690
Power [W]	277	208.7	285.64	268
Current [A]	19.4	11.5	19.3	18.1
g/W	6.50	5.87	5.78	6.31
18.5V				
configuration	no spec.	18.5V/13*6.5	18.5V/13*4	no spec.
Thrust [g]	no spec.	1640	2060	no spec.
Power [W]	no spec.	357.1	392.2	no spec.
Current [A]	no spec.	19.3	21.2	no spec.
g/W	no spec.	4.59	5.25	no spec
22.2V				
configuration	no spec.	22.2V/13*6.5	22.2V/13*4	22.2V/13*4.4
Thrust [g]	no spec.	2000	2800	3100
Power [W]	no spec.	548.3	628.3	737
Current [A]	no spec.	24.7	28.3	33.2
g/w	no spec.	3.65	4.46	4.21
Weight	97	113	131	188
kV	700	480	650	650
Price [euro]	89.9	44.9	38.16	72.9

Table 4.3: Overview of proper motor models

is not the best one because the thrust in each voltage supply class are the lowest. At this point the problem is, that the consumption of power does not increase linear with the thrust. If the motor has to work close to his upper performance limit, he consumes proportionally much more current than in a lower performance level. Furthermore the efficiency (admittedly, this is not the exact value but we just have the manufacturers parameters for our first estimation) is the lowest.

The T-Motors MT3515[TIG15] seems to be the most powerful in our selection (because we do not have any parameters from the U3 for 6S). The prize is okay but the model itself is very heavy, nearly double as heavy as the U3 model. We think that this motor is a little oversized for our purpose.

The Emax MT3515[EMA15] is very interesting. The performance characteristics seems to be proper and are specified for all supply voltage classes. Furthermore the prize is the lowest of all models. Because of good performance data and a low prize we decided to use this model

Aside from the practical measurements which are described further down, it is useful to evaluate the manufacturer's parameters theoretically. The estimation of the static thrust should be a good approximation to the given parameters. The following formulas are defined in[Sch15]

H/D is the proportion acclivity and diameter of the propeller.

$H/D) = \frac{4.4inch}{13inch} = \frac{4}{13}$

C is the coefficient for calculating the power.

$C_p = 0.0856 * (H/D) - 0.0091 = \frac{0,.0856*4}{13} - 0.0091 = 0.017238$

With this two values the consumed power P can be calculated.

$P = C_P * \rho * (\frac{n}{60})^3 * D^5 = 0.017238 * 1.24 * (\frac{4400}{60})^3 * 0.3302^5 = 33.09$

Now with this data we can calculate the thrust S which is generated by one propeller.

$S = 0.67 * (\frac{\rho}{2} * \pi * D^2 * P)^{\frac{1}{3}} = 0.67 * (\frac{1.24}{2} * \pi * 0.3302^2 * 33.09^2)^{\frac{1}{3}} = 4.12$

So the generated thrust is about 4.12N which are 412g. This equates nearly perfect the parameters which are given by the manufacturer (410g)[EMA15].

The Propeller

The choice of the best possible propeller is very important for an adequate flight time and behaviour. A propeller is basically defined by three values:

- left/right
- caliber
- acclivity

A propeller is driven by a motor, fixed on the body of the model. While turning the propeller, the motor generates angular momentum in the opposite direction. In technical issues this is called the yaw moment. For balancing this and preventing a rotation of the body there are basically two solutions:

- Using a tail propeller which generates counter-pressure against the yaw moment of the main propeller. This technique is used in helicopters. For our hexacopter this is not a usable solution because the technical realization is very complex.

- Using an equal number of propellers, so that each right-turning propeller will be compensated by a left-turning one. In this solution the propellers compensate their yaw moments to a sum of zero. This is the configuration which we will use for our system.

The regulation of the rotational speed of some propellers is the only way to influence the flight stabilization and attitude. The speedup of a motor lets it generate more boost and tip over the model. The boost of the system is not anymore just directed to the bottom but also in horizontal direction. This fact lets the model fly straight forward (or to the left or to the right, this depends on which motor is accelerated). Left and right flight operate the same way.

Therefore the propellers have to comply some more requirements:

- The motors permanently change their rotation speed, so the propellers have to be lightweight. Increasing the rotation speed of heavier propellers consumes more power and also takes more time.
- The propellers have to be balanced. Failing this, the imbalance will generate vibrations which disturb the sensor technology and the cameras. Some propellers are already balanced by the manufacturer, otherwise we would have to do this on our own.
- For testing it is suitable to use propellers which are a little bit softer, so they will not break directly when a crash occurs. In the final model it is more reasonable to use tighter propellers. The reason for this is that the blade tips of a soft propeller begin to pulsate. The pulsating part of the propeller is not generating boost anymore, so the flight-time will decline due to a lesser efficiency of the propellers.
- For bigger multicopters there are normally used CFK or APC propellers. These types are very robust.

For many motor models specific propellers are recommended. Normally these propellers are proper and fit on the motor shaft. So we did not specify a propeller model on our own. For the testing configuration, we will order the propellers that are recommended by the manufacturers. In a further work on the endconfiguration it is perhaps useful to optimize the used propeller model. Further tests for the evaluation of different models will be necessary.

Definition of Accumulator Weight

It is recommended to use the half of the maximum boost of the motors for the calculation of the weight. In reference to [EMA15] the half of the maximum consumed power is about 14A. The generated boost is about 1720g.

$$m_{accumulator} = \frac{I_{maxboost}}{2} * n_{motor} - m_{model} = 1720g * 6 - 4886g = 5434g$$

For a 50% load of the motors we are able to add about 5.6 kg accumulator weight. This is a theoretical calculation and there are a lot of external influences like barriers in the airflow or higher power consumption caused by component heating, which reduce the effective boost power of the motors. In reality the half of the effective maximum boost is smaller. Therefore we limited the whole models weight to 9 kg. This results in a weight of 4.11 kg for the accumulators.

Practical Measurements

The values given by the manufacturer seemed to be very optimistic. For that we wanted to evaluate the boost of one motor in an own practical measurement. In a mechanical installation we wanted to test the maximal boost of one motor and evaluate, how strong the affect of some obstacles in the air catchment area or air output is.

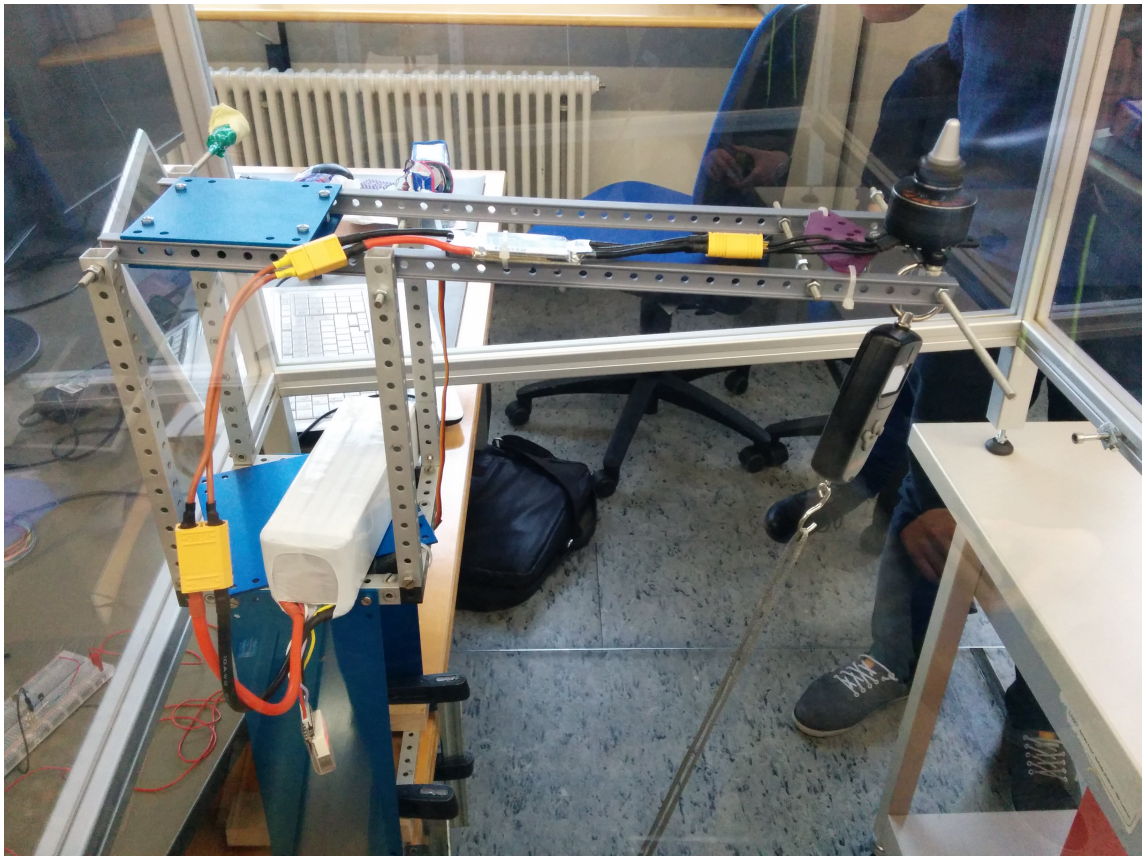


Figure 4.9: Testing platform for evaluate the maximum motor power

In illustration 4.9 you can see our test configuration. The fixture for the motor is not static, so that the generated boost normally causes a movement upsides. A tensile force analyzer is fixed at the motor and the ground, so that the generated boost can be measured. With this platform we were able to do two measurements:

- we could measure the maximum boost by trigger the ESC with 100%. The result of this measurement is enjoyable because the motor is able to generate a maximum boost of about 2720g. If we consider the weight of the fixture, the motor generates a boost of 3010g. This agrees nearly perfect with the value given by the manufacturer.
- we could measure, how long a constant boost can be generated with one full-loaded accumulator. For this measurements we triggered the motor with 50%. Furthermore we used an accumulator with a capacity of 5000mAh. The result of this measurement is enjoyable as well. Till the accumulator was empty, the motor was able to generate a constant boost of about 1400g for approximately 21 minutes. With a small calculation

  $$flighttime = \frac{flighttime_{measured} * multiple_{accumulator}}{n_{motor}} = \frac{23min * 4}{6} = 15.3min$$

  we have an estimation for the static flight-time if we use an overall accumulator capacity of 20000mAh. This seems to be a proper value for a first configuration. In

further projects there will surely be a lot of possibilites to optimize the mechanical construction of the model, the power supply system or something else, so that the flight time will increase.

4.2.4.2 ESC

The ESC is responsible to generate a tri-phase alternating current out of the given direct current from the accumulators. In illustration 4.10 you can see how an ESC is connected to the motor. For power supply the ESC is directly connected to an accumulator or indirectly over a power distribution platform (this is a useful solution if more than one motor or accumulator is used). The controller controls the supplied voltage to the motor by a PWM signal. This signal defines the motors speed by the duty cycle in the PWM signal. Furthermore the ESC is able to deliver a supply voltage to the controller, if required. This feature is called BEC, but it is not necessary for our platform because the flight regulation has a separated power supply.

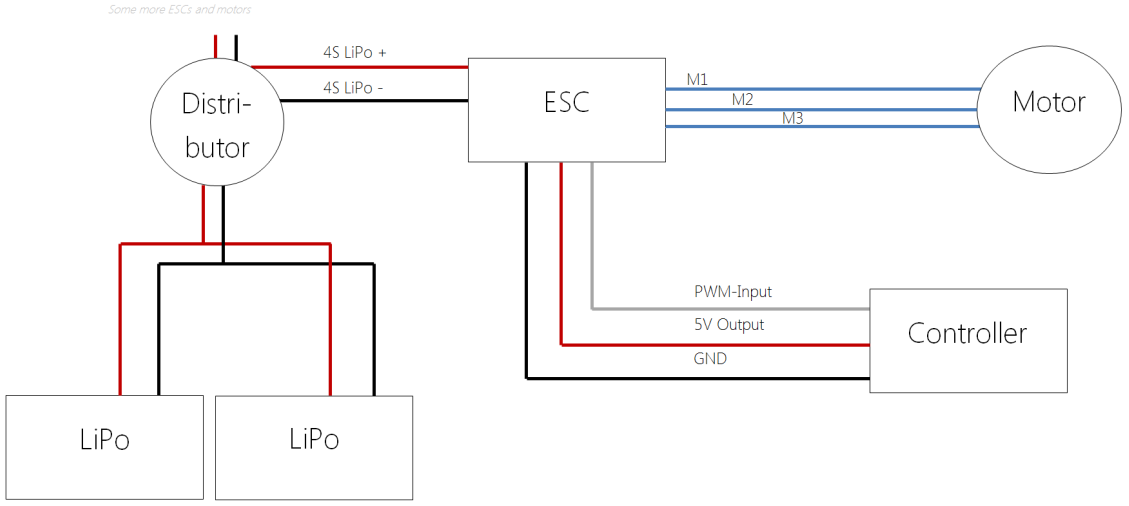


Figure 4.10: Schematic of the use of the ESC

The ESC for our model has to comply different requirements:

- The ESC has to work with a voltage level of 22.2V (6S)
- Due to a more stable flight and an higher precision of flight-controlling the ESC should work with the SimonK firmware:
  - Processing of 490KHz PWM signal. Ordinary ESC firmware can not work with such a high frequency.
  - Regulation of the motor with 18KHz. This prevents the annoying whistling from the motors (which is generating by 8KHz regulation). Furthermore the regulation is much more exact.
  - Support of 11 bit PWM signals
- The ESC has to handle a maximum current of 28.2A constantly. Because of some bursts it will be fine to have some tolerance to this value.
- The ESC should be as small as possible to be mounted at the holding construction for the motor. Furthermore the airflow must not be disturbed heavily by the mechanical size of the ESC.

- It is sufficient that the ESC is just cooled passively by the airflow which is generated by the propeller.
- Because there is no experience with ESCs in the project team and there are an high amount of ESCs available on the internet, the first order of ESCs is for testing purposes. Thus the ESC should not be too expensive.

For the first version of the model we decide to use the Bulltec ESC, available on [ESC15]. This component has a maximum constant current of 40A and uses the SimonK firmware. Furthermore it supports voltage systems from 2 to 6 cells and is not too expensive.

4.3 SoPC

For further information have a look on[Moh14][Chu12]

4.3.1 Version History

NIOS_MCAPI_Base_v01

The first version of the NIOS_MCAPI_Base consists basically of a HPS (hps_0) and two NIOS-II processors (cpu_s0 and cpu_s1) which are connected via FIFO bridges (fifo_bridge_cpuM_cpus0, fifo_bridge_cpuM_cpus1, fifo_bridge_cpus0_cpus1). To avoid mismatches between hardware and software configuration a system ID peripheral component (sysid_qsys) is included. The clock signals are generated by two PLLs. The first PLL (system_pll) generates a 125 MHz clock signal for both NIOS-II processors and a 25MHz clock signal for their components while the second PLL (sdram_pll) generates a 143 MHz signal to clock the SDRAM. To access their components, each NIOS-II processor is connected to a clock crossing bridge (s0_io_clockCrossing_bridge, s1_io_clockCrossing_bridge). The first NIOS-II is using the SDRAM as instruction memory and therefor it is connected with the SDRAM controller (sdram) over the second clock crossing bridge (sdram_clockCrossing_bridge). Over the first clock crossing bridge, the processor is connected to a timer (timer_cpu_s0), a JTAG UART interface (jtag_uart_cpu_s0) and a PIO interface (pio_aliveTest_cpu_s0). The second Nios-II is connected to a timer (timer_cpu_s1), a JTAG UART interface (jtag_uart_cpu_s1), a PIO interface (pio_aliveTest_cpu_s1) and the On-Chip memory (onchip_sram). The JTAG UART interface is used to load and debug programs on the Nios-II. Each PIO interfaces is connected to two red LEDs and mainly used for debug purposes.

NIOS_MCAPI_Base_v02

In the second version an I2C master is connected to the Nios-II (cpu_s0) processor via the Clock Crossing Bridge (s0_io_clockCrossing_bridge).

NIOS_MCAPI_Base_v03

In the third version the interrupt numbers from the FIFO bridges to the HPS are changed. Now fifo_bridge_cpuM_cpus0 has IRQ number 2 and fifo_bridge_cpuM_cpus1 has IRQ number 3. Also the interrupts are now connected to the lower 32 Bit interrupt line of the ARM.

NIOS_MCAPI_Base_v04

In the fourth version a second I2C master is included. This component is connected with the second Nios-II (cpu_s1) processor via the Clock Crossing Bridge (s1_io_clockCrossing_bridge).

NIOS_MCAPI_Base_v05

— NO DIFFERENCE —

NIOS_MCAPI_Base_v06

In the sixths version a PWM unit is connected to the Nios-II (cpu_s0). Also a PLL for the PWM is added. The PLL generates a signal of 1.505199 MHz.

NIOS_MCAPI_Base_v07

In the seventhsversion seven additional PWM units are connected to the NIOS-II (cpu_s0).

4.3.2 Components of the latest SoPC version

Component	Description
clk_0	Main clock source used by all PLLs as reference clock.
system_pll	PLL to generate a signal of 125 MHz used as clock signal for both Nios-II processors. Furthermore a second signal of 25 MHz is generated for all connected components.
sdram_pll	PLL to generate a signal of 143 MHz used as clock signal for the SDRAM controller.
pwm_pll	PLL to generate a signal of 1.505199 MHz used as clock signal for the PWM modules.

Table 4.4: Clock signal sources

Component	Description
hps_0	Interface to the HPS. In our case the HPS is a dual-core ARM Cortex-9 MPCore processor at 925 MHz.
intr_capturer_0	Interrupt capturer to identify the interrupt source. In our SoPC this component is not in use.
hps_only_master	Needed to allow the HPS to use the JTAG channel.
fpga_only_master	Needed to allow the FPGA to use the JTAG channel.
sysid_qsys	Identifies the SoPC. Needed to avoid mismatches between hardware and software configuration.

Table 4.5: HPS interface and general components

Component	Description
fifo_bridge_cpuM_cpus0	FIFO bridge between HPS and Nios-II processor for flight control. The FIFO has a capacity of 32x 4Byte in each direction. The bridge is used by the MCAPI as receive buffer.
fifo_bridge_cpuM_cpus1	FIFO bridge between HPS and Nios-II processor for collision detection. The FIFO has a capacity of 32x 4Byte in each direction. The bridge is used by the MCAPI as receive buffer.
fifo_bridge_cpus0_cpus1	FIFO bridge between both Nios-II processors. The FIFO has a capacity of 32x 4Byte in each direction. The bridge is used by the MCAPI as receive buffer.

Table 4.6: FIFO bridges

Component	Description
cpu_s0	Nios-II processor running at 125 MHz clock speed. As instruction memory the external 64 MByte SDRAM is used. This processor is responsible for the flight control of the multicopter.
s0_io_clockCrossing_bridge	Bridge to allow the higher clocked Nios-II processor to access lower clocked components.
timer_cpu_s0	Interval timer for the Nios-II processor.
i2c_cpu_s0	I2C master component for the Nios-II. Makes it possible to connect I2C slaves with the Nios-II.
jtag_uart_cpu_s0	JTAG UART interface to program and debug the Nios-II processor via USB-Blaster II.
pio_aliveTest_cpu_s0	PIO unit connected with two red LEDs. Only used for debugging purposes.
pwm_cpu_s0_[1:8]	PWM units to generate PWM signals to regulate the speed of each motor separately.
sdram_clockCrossing_bridge	Bridge to allow the lower clocked Nios-II processor to access the higher clocked SDRAM controller.
sdram	SDRAM controller which connects the external SDRAM with the FPGA.

Table 4.7: Flight control components

Component	Description
cpu_s1	Nios-II processor running at 125 MHz clock speed. As instruction memory the 352 kByte large On-Chip SRAM.
s1_io_clockCrossing_bridge	Bridge to allow the higher clocked Nios-II processor to access lower clocked components.
timer_cpu_s1	Interval timer for the Nios-II processor.
i2c_cpu_s1	I2C master component for the Nios-II. Makes it possible to connect I2C slaves with the Nios-II.
jtag_uart_cpu_s1	JTAG UART interface to program and debug the Nios-II processor via USB-Blaster II.
pio_aliveTest_cpu_s1	PIO unit connected with two red LEDs. Only used for debugging purposes.
onchip_sram	On-Chip SRAM with a total capacity of 352 kByte.

Table 4.8: Collision detection components

4.3.3 PWM Controller

The PWM controller was not used in the standard version. It was modified by the project team in order to reach an appropriate clock frequency.
The ESCs need an input frequency of 490Hz which has to be generated by the PWM controller. The output frequency of the PWM controller is influenced by two factors:

- The input clock frequency
- The divider value in the VHDL code

The input clock frequency is set to 1.505199 MHz. The VHDL code contains a divider value to adjust the input frequency. It divides the clock frequency by x*255. Here x is the

divider value. The output frequency can be calculated with the following formula:

$F_{out}(x) = \frac{F_{input}}{x*255}$

- F_{input} is the input frequency
- x is the divider value

If you solve the equation for x you get the following term:

$x = \frac{\frac{F_{input}}{F_{out}}}{255} = \frac{\frac{1505199Hz}{490Hz}}{255} = 12.05$

As divider value we can just set integers so we set it to 12. The specific part of code that has to be adjusted is contained in pwm.vhd and is as follows:

```
1 if divider = "0000001100" then
  --toggle signal
3 else
  --divider=
5 end if
```

4.3.4 SoPC Block Diagram

In illustration 4.11 you can see the diagram of the latest SoPC version.

4.4 Software

4.4.1 Communication Concept

Scenarios

Software

The system has to do three main tasks that are distributed on the three processors.
The task with the highest priority is done by one of the Nios II processors. This program has to keep the hexacopter in the air stable. The other Nios II processor realizes a collision detection while the ARM processor reads data from two Kinect cameras to create a map of the environment and check what is the current position of the hexacopter.
In case of that a system is connected via Ethernet, the extending system takes the ARM processors tasks.

Illustration 4.12 shows which communication messages have to be exchanged between the three processors.

The flight control is the center part of the system. It reads the position sensors like a gyroscope and an accelerometer and has to transmit this data to the ARM processor and the collision detection.
The algorithms, that are running on the ARM processor, have to know this data for their calculations.
The collision detection has to know this data to be able to interpret the data of the collision sensors right.
The two other processors both have to be able to send control commands to the flight control.
The software, that runs on the ARM processor, has to control the system to fly along a path that was calculated by the software. The collision detection has to be able to control the system to avoid a collision with any object.

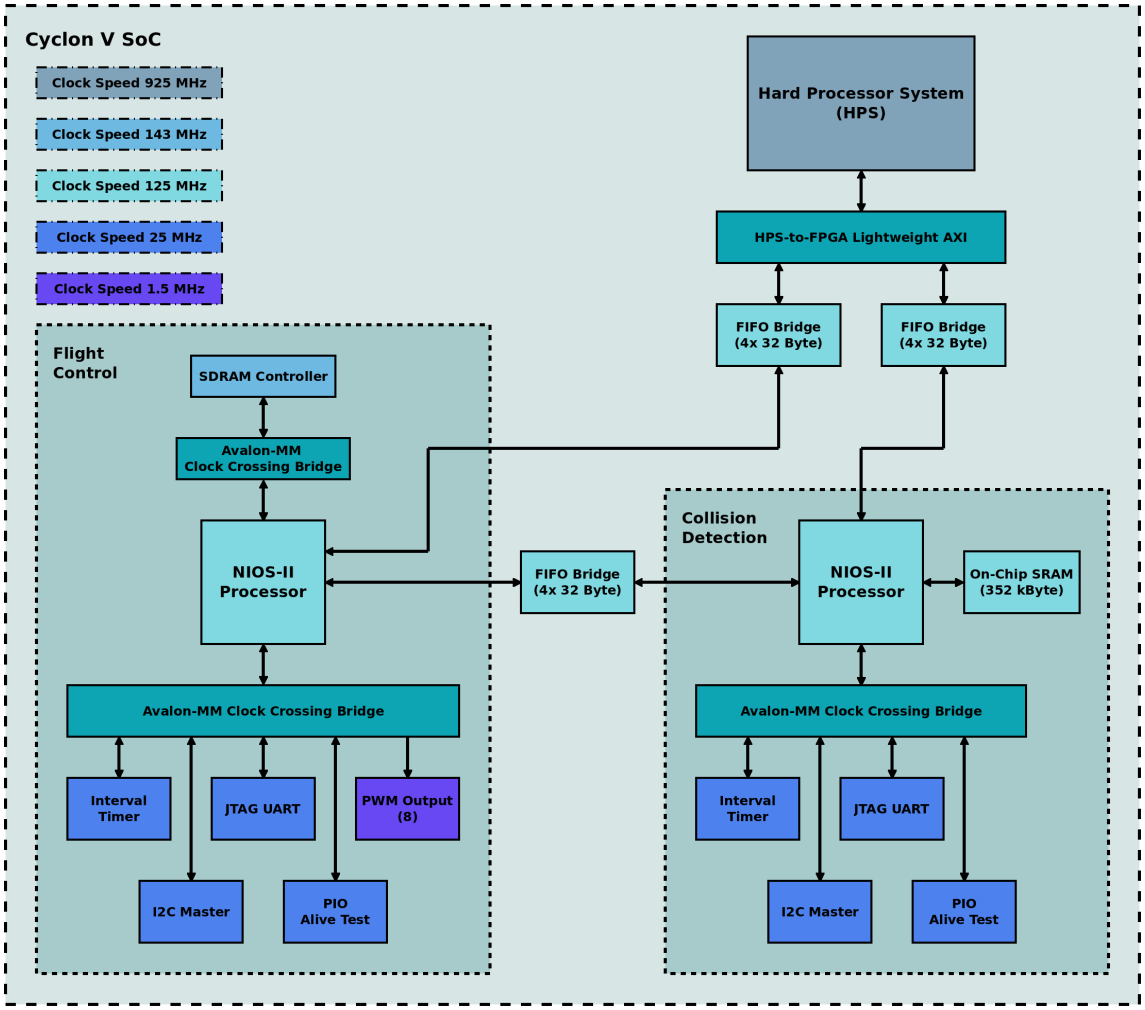


Figure 4.11: Latest version of the SoPC

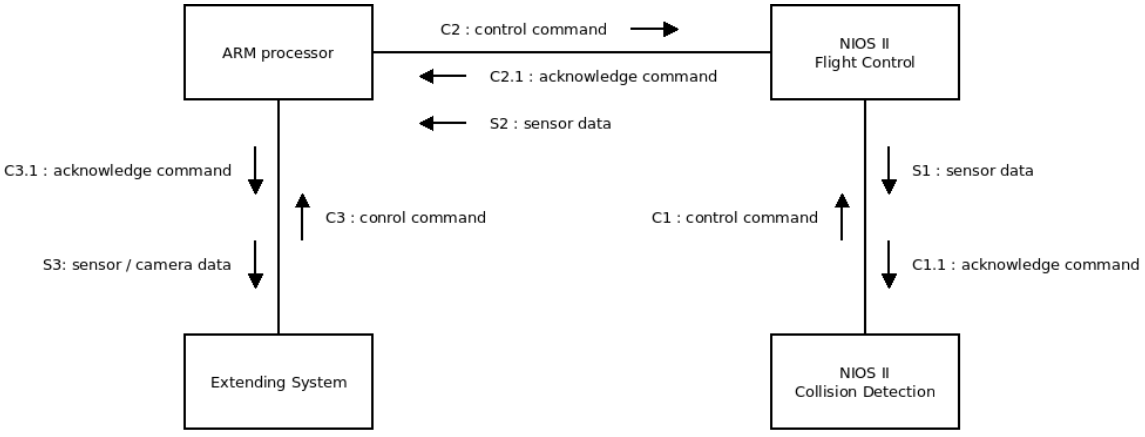


Figure 4.12: Latest version of the SoPC

Name	Description	Command
Up	Moves the aircraft up with a specified intensity, way or duration.	
Down	Moves the aircraft down with a specified intensity, way or duration.	
Fly-Direction	Moves the aircraft in a specific direction with a specified intensity, direction angle, way or duration.	
Forward	Moves the aircraft forward with a specified intensity, way or duration.	
Backward	Moves the aircraft backward with a specified intensity, way or duration.	
Left	Moves the aircraft left with a specified intensity, way or duration.	
Right	Moves the aircraft right with a specified intensity, way or duration.	
Turn Left	Turns the aircraft left with a specified intensity, angle or duration.	
Turn Right	Turns the aircraft right with a specified intensity, angle or duration.	
Stop	Interrupts the current commands so that the aircraft stays in hovering flight.	

Table 4.9: Concept for flight control

Hardware

The existing multiprocessor system consists of three processors.
One processor is a Dual ARM Cortex A9 processor with a clock frequency of 800MHz, while the other two processors are Nios II soft processors which have a clock frequency of 50MHz.
All three processors are connected via FIFO bridges. Between each two processors there is one FIFO bridge that connects them. One FIFO bridge is able to save 32 32Bit data words. That means it has a maximum capacity of 128Bytes.
The whole system can be extended by another computer system, that is connected via Ethernet.
In this case the ARM processor acts as a system that forwards data between the extending system and the other processors. Additionally the ARM processor has to pass camera data to the extending system.

Communication Interface

Flight Control Interface

For controlling the aircraft from other system parts than the flight control. We defined an application-layer protocol that can be used to take influence on the flight control by sending control commands.

Table 4.9 shows a list of actions that have to be supported by the protocol.

These commands are sent to the fight control that executes the specific actions. Because there is more than one system that could send control commands to the flight control, a priority of control commands has to be defined. Therefore each system has a fixed priority,

whereby the collision detection should have the highest. The advantages of this are that no priority information has to be sent in the control commands and that no other system can manipulate its own priority.
Otherwise it would be possible that a system gives itself a higher priority than important systems such as the collision detection.

Sensor Data Interface

The ARM processor and the collision detection need sensor data that is read by the flight control. The ARM processor needs data of for example the gyroscope that is needed by the mapping algorithm. The collision detection needs data of the flight control to decide which commands should be sent to avoid a collision without causing trouble through an overreaction.

Realization

Introduction to MCAPI

The MCAPI provides an API for the communication between processor cores in embedded systems. It offers different possibilities to send data to another system. One way is to send data connection-less which is not used in our system. The other way is to use a connection-based method which we use in our system. There are also various ways of using connection-based communication via the MCAPI. One possibility to send so called packets the other way is to send scalars. By using the first method the data is sent in a packet that has a variable size that is variable.
By using the second method the data that can be send as scalars of 8-bit, 16-bit, 32-bit or 64-bit.

Control Command Protocol

Data format for data exchange
One control command is sent in one packet.

Structure of one control packet:

4-bit Command | 7-bit Intensity | 12-bit Duration | 9-bit Angle
- 32-bit
A command has a maximum size of 32-bits of data without overhead that is caused by use of the MCAPI.
The fields of the command have a fixed size and are optional. For a list of commands and what parameters are required see 2.1.

Address formats for data exchange
An address format is not needed here because the mapping is defined static in a mapping file.

Address mapping
As said before the mapping of the endpoints is done in the mapping file mapping.h.

Routing
In our system no routing is needed. There are no other devices between two endpoints.

They're connected directly via one FIFO.

Detection of transmission errors
A detection of transmission errors is not needed here. The communication channel provides a very reliable way of exchanging data. The connection between two processors is very short and is not made via a longer cable. A point that makes the communication through the FIFO also very reliable is, that this is a point to point connection between two processors and no other processors communicate over the same connection. This would be the case for example in a bus system.

Acknowledgments
After the execution of a command, the correlated system will be informed that his command has been executed.

Loss of information
For the prevention of the loss of information you could define timeouts and retries that would be made after a timeout occurred. In our case there are no timeouts defined.

Direction of transmission flow
Because a MCAPI channel is unidirectional, there have to be two channels between two nodes to exchange data in both directions. The channel going to the flight control is used for the commands while the channel coming from the flight control is used for the acknowledgements.

Flow control
The flow control is managed by the MCAPI. If the send buffer is full no more information can be sent

Sensor Data Protocol

Data format for data exchange

Address formats for data exchange
An address format is not needed here because the mapping is defined static in a mapping file.

Address mapping
As said before the mapping of the endpoints is done in the mapping file mapping.h.

Routing
In our system no routing is needed. There are no other devices between two endpoints. They're connected directly via one FIFO.

Detection of transmission errors
A detection of transmission errors is not needed here. The communication channel provides a very reliable way of exchanging data. The connection between two processors is very short and is not made via a longer cable. A point that makes the communication

through the FIFO also very reliable is, that this is a point to point connection between two processors and no other processors communicate over the same connection. This would be the case for example in a bus system.

Acknowledgments
In our protocol no acknowledgments are needed. You can control if a data word is queued in the FIFO for the communication partner by reading the send level register.

Loss of information
For the prevention of the loss of information you could define timeouts and retries that would be made after a timeout occurred. In our case there are no timeouts defined.

Direction of transmission flow
For the sensor data protocol a simplex connection is used, because there is no need to send data back after receiving sensor data. Therefore a single MCAPI channel coming from the flight control is sufficient.

Flow control
The flow control is managed by the MCAPI. If the send buffer is full no more information can be sent.

Realization with the MCAPI

Control Command Protocol
For the realization of the Control Command Protocol the scalar functions of the MCAPI are used, whereby each command is sent as a 32-bit scalar. It would seem the thing to do, because a control command is at maximum 32-bit large.

Sensor data protocol
For the realization of the Sensor Data Protocol the packet functions of the MCAPI are used. This way it is possible to send multiple sensor data as a bundle in only one MCAPI packet. The overhead generated by the MCAPI should be less than sending each sensor data as a MCAPI scalar message.

Sensorlist

- 1 Gyroscope
- 1 Accelerometer
- 4 Distance Sensors
- 2 Kinect Cameras
  - 2 RGB
  - 2 Depth Sensors

The sensors of the flight control are polled by single tasks and the sensor values are stored. For sending the stored sensor data there are separate tasks in order to prevent the system from blocking that would be caused by the blocking scalar send function..

The sensors of the collision detection are polled by a task but no sensor data is sent

directly, because the system decides for appropriate control commands and sends them to the flight control. For an overview of available functions of the MCAPI see the Multicore Communication API reference in the next section.

Mulitcore Communications API reference

Data types

Data Type	Description
mcapi_domain_t	MCAPI domain
mcapi_node_t	MCAPI node
mcapi_port_t	MCAPI port
mcapi_endpoint_t	MCAPI endpoint
mcapi_pktchan_send_hndl_t	Sending handle to connected packet channel
mcapi_pktchan_recv_hndl_t	Receiving handle from connected packet channel
mcapi_sclchan_send_hndl_t	Sending handle to connected scalar channel
mcapi_sclchan_recv_hndl_t	Receiving handle from connected scalar channel
mcapi_uint8|16|32|64_t	Unsigned 8-, 16-, 32- and 64-bit scalar
mcapi_int8|16|32|64_t	Singed 8-, 16-, 32- and 64-bit scalar
mcapi_boolean_t	Boolean
mcapi_priority_t	Priority type
mcapi_request_t	State of a pending non-blocking MCAPI transaction
mcapi_status_t	Error or status code
mcapi_timeout_t	Duration mcapi_wait_any will block

Table 4.10: Data Types

Error and Status Codes

Error or Status Code	Description
MCAPI_SUCCESS	Successful operation
MCAPI_PENDING	Pending operation without errors
MCAPI_TIMEOUT	Operation timed out
MCAPI_ERR_PARAMETER	Incorrect parameter
MCAPI_ERR_DOMAIN_INVALID	Parameter is an invalid domain
MCAPI_ERR_NODE_INVALID	Parameter is an invalid node
MCAPI_ERR_NODE_INITFAILED	Could not initialize node
MCAPI_ERR_NODE_INITIALIZED	Node is already initialized
MCAPI_ERR_NODE_NOTINIT	Node is not initialized
MCAPI_ERR_NODE_FINALFAILED	Node could not be finalized
MCAPI_ERR_PORT_INVALID	Parameter is an invalid port
MCAPI_ERR_ENDP_INVALID	Parameter is an invalid endpoint descriptor
MCAPI_ERR_ENDP_EXISTS	Endpoint already exists
MCAPI_ERR_ENDP_NOTOWNER	Endpoint can only be deleted by his owner
MCAPI_ERR_ENDP_REMOTE	Operation not allowed on remote node endpoint
MCAPI_ERR_ATTR_INCOMPATIBLE	Connection of endpoints with incompatible attributes not possible
MCAPI_ERR_ATTR_SIZE	Attribute size is incorrect
MCAPI_ERR_ATTR_NUM	Attribute number is incorrect
MCAPI_ERR_ATTR_VALUE	Attribute value is incorrect
MCAPI_ERR_ATTR_NOTSUPPORTED	Attribute is not supported
MCAPI_ERR_ATTR_READONLY	Attribute is read-only
MCAPI_ERR_MSG_LIMIT	Message size exceeds maximum size
MCAPI_ERR_MSG_TRUNCATED	Message size exceeds buffer size
MCAPI_ERR_CHAN_OPEN	Channel is open, certain operations not allowed
MCAPI_ERR_CHAN_TYPE	Channel type is different from channel type on endpoint
MCAPI_ERR_CHAN_DIRECTION	Attempt to open a send handle on a port connected as a receiver or vice versa
MCAPI_ERR_CHAN_CONNECTED	Channel is already connected
MCAPI_ERR_CHAN_OPENPENDING	An open request is pending
MCAPI_ERR_CHAN_CLOSEPENDING	An close request is pending
MCAPI_ERR_CHAN_NOTOPEN	Channel not open or can not be closed
MCAPI_ERR_CHAN_INVALID	Parameter is an invalid channel handle
MCAPI_ERR_PKT_SIZE	Packet size exceed maximum size
MCAPI_ERR_TRANSMISSION	Transmission failure
MCAPI_ERR_PRIORITY	Priority level is incorrect
MCAPI_ERR_BUF_INVALID	Buffer descriptor is invalid
MCAPI_ERR_MEM_LIMIT	Out of memory
MCAPI_ERR_REQUEST_INVALID	Parameter is an invalid request handle
MCAPI_ERR_REQUEST_LIMIT	Out of request handles
MCAPI_ERR_REQUEST_CANCELLED	Request was already canceled
MCAPI_ERR_WAIT_PENDING	A wait is pending
MCAPI_ERR_GENERAL	Other error conditions
MCAPI_STATUSCODE_END	This should always be last

Table 4.11: Error Status Codes

General Functions

Initialize MCAPI environment

```
void mcapi_initialize(
  MCAPI_IN mcapi_domain_t domain_id,
  MCAPI_IN mcapi_node_t node_id,
  MCAPI_IN mcapi_node_attributes_t* mcapi_node_attributes,
  MCAPI_IN mcapi_param_t* mcapi_parameters,
  MCAPI_OUT mcapi_info_t* mcapi_info,
  MCAPI_OUT mcapi_status_t* mcapi_status);
```

Finalize MCAPI enviroment

```
void mcapi_finalize(
  MCAPI_OUT mcapi_status_t* mcapi_status);
```

Get domain identifier

```
mcapi_domain_t mcapi_domain_id_get(
  MCAPI_OUT mcapi_status_t* mcapi_status);
```

Get node identifier

```
mcapi_node_t mcapi_node_id_get(
  MCAPI_OUT mcapi_status_t* mcapi_status);
```

Initialize structure values

```
void mcapi_node_init_attributes(
  MCAPI_OUT mcapi_node_attributes_t* mcapi_node_attributes,
  MCAPI_OUT mcapi_status_t* mcapi_status);
```

Change default values

```
void mcapi_node_set_attribute(
  MCAPI_OUT mcapi_node_attributes_t* mcapi_node_attributes,
  MCAPI_IN mcapi_uint_t attribute_num,
  MCAPI_IN void* attribute,
  MCAPI_IN size_t attribute_size,
  MCAPI_OUT mcapi_status_t* mcapi_status);
```

Query node attributes

```
void mcapi_node_get_attribute(
  MCAPI_IN mcapi_domain_t domain_id,
  MCAPI_IN mcapi_node_t node_id,
  MCAPI_IN mcapi_uint_t attribute_num,
  MCAPI_OUT void* attribute,
  MCAPI_IN size_t attribute_size,
  MCAPI_OUT mcapi_status_t* mcapi_status);
```

Endpoint Functions

Create endpoint

```
mcapi_endpoint_t mcapi_endpoint_create(
  MCAPI_IN mcapi_port_t port_id,
  MCAPI_OUT mcapi_status_t* mcapi_status);
```

Delete endpoint

```
void mcapi_endpoint_delete(
  MCAPI_IN mcapi_endpoint_t endpoint,
  MCAPI_OUT mcapi_status_t* mcapi_status);
```

Get endpoint identifier

```
//Non-blocking version
void mcapi_endpoint_get_i(
  MCAPI_IN mcapi_domain_t domain_id,
  MCAPI_IN mcapi_node_t node_id,
  MCAPI_IN mcapi_port_t port_id,
  MCAPI_OUT mcapi_endpoint_t* endpoint,
  MCAPI_OUT mcapi_request_t* request,
  MCAPI_OUT mcapi_status_t* mcapi_status);


//Blocking Version
void mcapi_endpoint_get(
  MCAPI_IN mcapi_domain_t domain_id,
  MCAPI_IN mcapi_node_t node_id,
  MCAPI_IN mcapi_port_t port_id,
  MCAPI_OUT mcapi_timeout_t timeout,
  MCAPI_OUT mcapi_status_t* mcapi_status);
```

Query endpoint attributes

```
void mcapi_endpoint_get_attribute(
  MCAPI_IN mcapi_endpoint_t endpoint,
  MCAPI_IN mcapi_uint_t attribute_num,
  MCAPI_OUT void* attribute,
  MCAPI_IN size_t attribute_size,
  MCAPI_OUT mcapi_status_t* mcapi_status);
```

Assign endpoint attribute

```
void mcapi_endpoint_set_attribute(
  MCAPI_IN mcapi_endpoint_t endpoint,
  MCAPI_IN mcapi_uint_t attribute_num,
  MCAPI_IN void* attribute,
  MCAPI_IN size_t attribute_size,
  MCAPI_OUT mcapi_status_t* mcapi_status);
```

Message Functions

Send message

```
//Non-blocking version
void mcapi_msg_send_i(
  MCAPI_IN mcapi_endpoint_t send_endpoint,
  MCAPI_IN mcapi_endpoint_t receive_endpoint,
  MCAPI_IN void* buffer,
  MCAPI_IN size_t buffer_size,
  MCAPI_IN mcapi_priority_t priority,
  MCAPI_OUT mcapi_request_t* request,
  MCAPI_OUT mcapi_status_t* mcapi_status);
//Blocking version
void mcapi_msg_send(
  MCAPI_IN mcapi_endpoint_t send_endpoint,
  MCAPI_IN mcapi_endpoint_t receive_endpoint,
  MCAPI_IN void* buffer,
  MCAPI_IN size_t buffer_size,
  MCAPI_IN mcapi_priority_t priority,
  MCAPI_OUT mcapi_status_t* mcapi_status);
```

Receive message

```
//Non-blocking version
void mcapi_msg_recv_i(
  MCAPI_IN mcapi_endpoint_t receive_endpoint,
  MCAPI_OUT void* buffer,
  MCAPI_IN size_t buffer_size,
  MCAPI_OUT mcapi_request_t* request,
  MCAPI_OUT mcapi_status_t* mcapi_status);

//Blocking version
void mcapi_msg_recv(
  MCAPI_IN mcapi_endpoint_t receive_endpoint,
  MCAPI_OUT void* buffer,
  MCAPI_IN size_t buffer_size,
  MCAPI_OUT size_t* received_size,
  MCAPI_OUT mcapi_status_t* mcapi_status);
```

Check message availability

```
mcapi_uint_t mcapi_msg_available(
  MCAPI_IN mcapi_endpoint_t receive_endpoint,
  MCAPI_OUT mcapi_status_t* mcapi_status);
```

Packet channel functions

Connect endpoints

```
void mcapi_pktchan_connect_i(
  MCAPI_IN mcapi_endpoint_t send_endpoint,
  MCAPI_IN mcapi_endpoint_t recevie_endpoint,
  MCAPI_OUT mcapi_request_t* request,
  MCAPI_OUT mcapi_status_t* mcapi_status);
```

Open receive side

```
void mcapi_pktchan_recv_open_i(
  MCAPLOUT mcapi_pktchan_recv_hndl_t* receive_handle,
  MCAPLIN mcapi_endpoint_t receive_endpoint,
  MCAPLOUT mcapi_request_t* request,
  MCAPLOUT mcapi_status_t* mcapi_status);
```

Open send side

```
void mcapi_pktchan_send_open_i(
  MCAPLOUT mcapi_pktchan_send_hndl_t* send_handle,
  MCAPLIN mcapi_endpoint_t send_endpoint,
  MCAPLOUT mcapi_request_t* request,
  MCAPLOUT mcapi_status_t* mcapi_status);
```

Close receive side

```
void mcapi_pktchan_recv_close_i(
  MCAPLIN mcapi_pktchan_recv_hndl_t receive_handle,
  MCAPLOUT mcapi_request_t* request,
  MCAPLOUT mcapi_status_t* mcapi_status);
```

Close send side

```
void mcapi_pktchan_send_close_i(
  MCAPLIN mcapi_pktchan_send_hndl_t send_handle,
  MCAPLOUT mcapi_request_t* request,
  MCAPLOUT mcapi_status_t* mcapi_status);
```

Send packet

```
//Non-blocking version
void mcapi_pktchan_send_i(
  MCAPLIN mcapi_pktchan_send_hndl_t send_handle,
  MCAPLIN void* buffer,
  MCAPLIN size_t buffer_size,
  MCAPLOUT mcapi_request_t* request,
  MCAPLOUT mcapi_status_t* mcapi_status);

//Blocking version
void mcapi_pktchan_send(
  MCAPLIN mcapi_pktchan_send_hndl_t send_handle,
  MCAPLIN void* buffer,
  MCAPLIN size_t buffer_size,
  MCAPLOUT mcapi_status_t* mcapi_status);
```

Receive packet

```
//Non-blocking version
void mcapi_pktchan_recv_i(
  MCAPLIN mcapi_pktchan_recv_hndl_t receive_handle,
  MCAPLOUT void** buffer,
  MCAPLOUT mcapi_request_t* request,
```

```
  MCAPLOUT mcapi_status_t* mcapi_status);

//Blocking version
void mcapi_pktchan_recv(
  MCAPLIN mcapi_pktchan_recv_hndl_t receive_handle,
  MCAPLOUT void** buffer,
  MCAPLOUT size_t* received_size,
  MCAPLOUT mcapi_status_t* mcapi_status);
```

Release buffer

```
void mcapi_pktchan_release(
  MCAPLIN void* buffer,
  MCAPLOUT mcapi_status_t* mcapi_status);
```

Check buffer release

```
mcapi_boolean_t mcapi_pktchan_release_test(
  MCAPLIN void* buffer,
  MCAPLOUT mcapi_status_t* mcapi_status);
```

Check packet availability

```
mcapi_uint_t mcapi_pktchan_available(
  MCAPLIN mcapi_pktchan_recv_hndl_t receive_handle,
  MCAPLOUT mcapi_status_t* mcapi_status);
```

Scalar Channel Functions

Connect endpoints

```
void mcapi_sclchan_connect_i(
  MCAPLIN mcapi_endpoint_t send_endpoint,
  MCAPLIN mcapi_endpoint_t receive_endpoint,
  MCAPLOUT mcapi_request_t* request,
  MCAPLOUT mcapi_status_t* mcapi_status);
```

Open receive side

```
oid mcapi_sclchan_recv_open_i(
  MCAPLOUT mcapi_sclchan_recv_hndl_t* receive_handle,
  MCAPLIN macpi_endpiont_t receive_endpoint,
  MCAPLOUT mcapi_request_t* request,
  MCAPLOUT mcapi_status_t* mcapi_status);
```

Open send side

```
oid mcapi_sclchan_send_open_i(
  MCAPLOUT mcapi_sclchan_send_hndl_t* send_handle,
  MCAPLIN mcapi_endpoint_t send_endpoint,
  MCAPLOUT mcapi_request_t* request,
  MCAPLOUT mcapi_status_t* mcapi_status);
```

Close receive side

```
void mcapi_sclchan_recv_close_i(
  MCAPI_IN mcapi_sclchan_recv_hndl_t receive_handle,
  MCAPI_OUT mcapi_request_t* request,
  MCAPI_OUT mcapi_status_t* mcapi_status);
```

Close send side

```
void mcapi_sclchan_send_close_i(
  MCAPI_IN mcapi_sclchan_send_hndl_t send_handle,
  MCAPI_OUT mcapi_request_t* request,
  MCAPI_OUT mcapi_status_t* mcapi_status);
```

Receive scalar

```
mcapi_uint{8|16|32|64}_t mcapi_sclchan_uint{8|16|32|64}(
  MCAPI_IN mcapi_sclchan_recv_hndl_t receive_handle,
  MCAPI_OUT mcapi_status_t* mcapi_status);
```

Send scalar

```
void mcapi_sclchan_send_uint{8|16|32|64}(
  MCAPI_IN mcapi_sclchan_send_hndl_t send_handle,
  MCAPI_IN mcapi_uint{8|16|32|64}_t dataword,
  MCAPI_OUT mcapi_status_t* mcapi_status);
```

Check scalar availability

```
mcapi_uint_t mcapi_sclchan_available(
  MCAPI_IN mcapi_sclchan_recv_hndl_t receive_handle,
  MCAPI_OUT mcapi_status_t* mcapi_status);
```

Non-blocking management functions

Wait for completion of operation

```
mcapi_boolean_t mcapi_wait(
  MCAPI_IN mcapi_request_t* request,
  MCAPI_OUT size_t* size,
  MCAPI_IN mcapi_timeout_t timeout,
  MCAPI_OUT mcapi_status_t* mcapi_status);
```

Wait for completion of operation list

```
mcapi_uint_t mcapi_wait_any(
  MCAPI_IN size_t number,
  MCAPI_IN mcapi_request_t* requests,
  MCAPI_OUT size_t* size,
  MCAPI_IN mcapi_timeout_t timeout,
  MCAPI_OUT mcapi_status_t* mcapi_status);
```

Check for non-blocking operation completion

```
mcapi_boolean_t mcapi_test(
  MCAPI_IN mcapi_request_t* request,
  MCAPI_OUT size_t* size,
  MCAPI_OUT mcapi_status_t* mcapi_status);
```

Cancel non-blocking operation

```
void mcapi_cancel(
  MCAPI_IN mcapi_request_t* request,
  MCAPI_OUT mcapi_status_t* mcapi_status);
```

Support Function

Display MCAPI status message

```
char* mcapi_display_status(
  MCAPI_IN mcapi_status_t mcapi_status,
  MCAPI_OUT char* status_message,
  MCAPI_IN size_t size);
```

4.4.2 Software Architecture

In this section it shall be described which software architecture was designed for the system. It hereby focuses on the software architecture of the flight control system. The design was visualized in an UML class diagram which can be seen in illustration 4.13.

The software architecture is segmented into different layers. The first layer contains the drivers for the controllers that are synthesized into the FPGA as IP-Cores.
The second layer contains the drivers for the sensors and actors that are connected to the controllers which are controlled via the first layer.
The third layer's purpose is to filter the input data that the system reads from the various sensors. This is an essential task because the aircraft is exposed to intensive vibrations during the flight.
The fourth layer basically abstracts the sensors and actors. It brings the sensors data to a more manageable state and offers an abstraction for the control of the motors. It just offers an interface to say in which direction the aircraft should fly and with which intensity. It then computes an output value for each motor controller.
The fifth layer is the main program contained in the Flight_Control module which runs the stabilization algorithm. This module takes commands from the Command_Receiver module that receives commands from different input sources which are basically the other systems and the RC receiver.
The Command_Receiver also has to check the commands priorities in order to execute the command with the highest priority before lower prioritized commands.
In the project just layer 1 and 2 could be implemented. That means most of the sensors can be read through the different drivers.
The driver implementation will be described in detail in the next section.

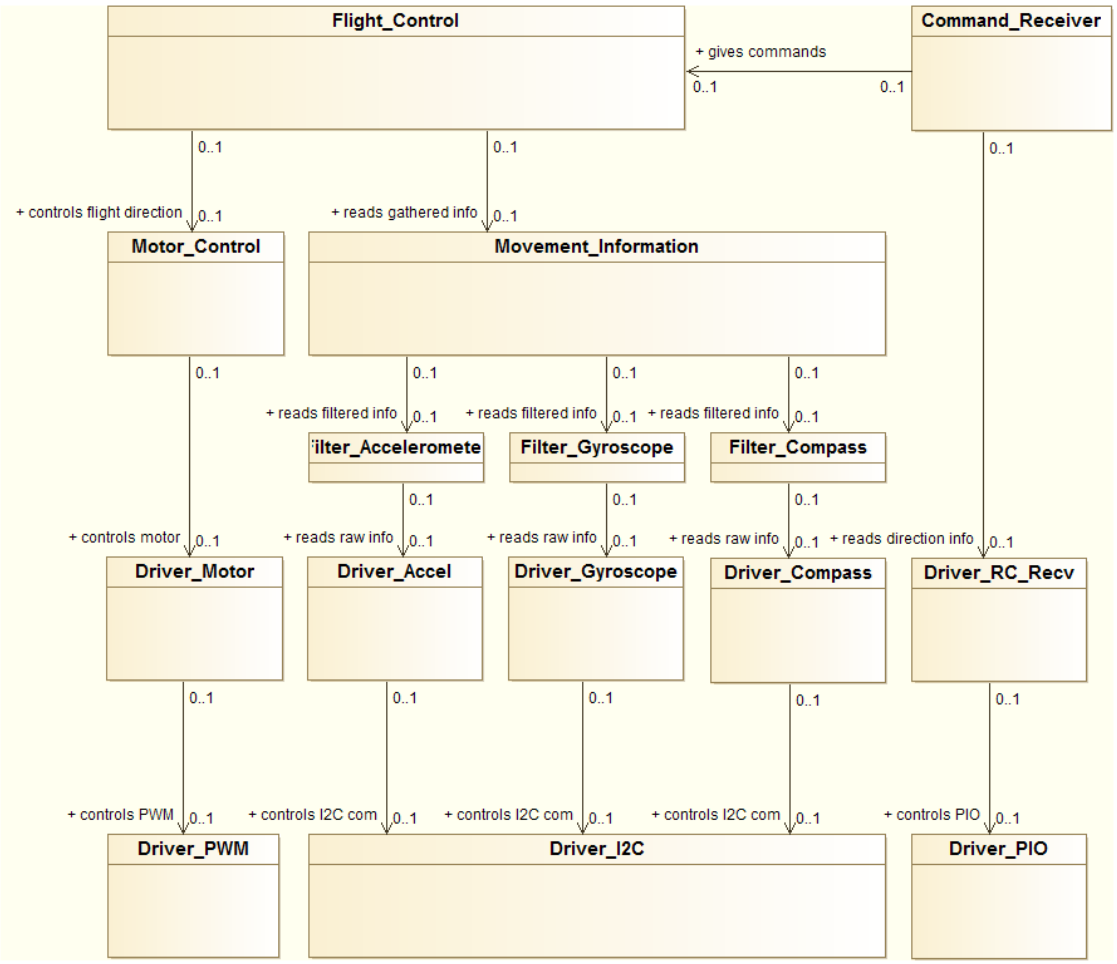


Figure 4.13: Software model for the flight control

4.4.3 Drivers

4.4.3.1 PWM Driver

This driver provides an interface to control the PWM signal generators. The driver is divided in a source file "'b_pwmdriver.c"' and a header file "'b_pwmdriver.h"', which can be found in the drivers folder.
At the moment 8 PWM controllers are realized through an IP-Core each and are supported by this driver. To configure the duty time of the PWM signal each IP-Core offers an 8-bit register, so the PWM signals have an 8-bit resolution. For more information see chapter 4.3.3.
For an easier readable and changeable code the base addresses of the signal width registers are mapped to names, so they can be easily used in the code. Also with this defines it is simple to modify them if the addresses of the registers are changed later on.
First an initialization method is provided by the driver. It sets all signals to the lowest signal width, which is zero. This method must be run at the start of the system to ensure no other PWM signal is generated by the controllers due to false values are in the signal width registers.
Furthermore two different functions to set the duty cycle of the PWM signal exist. Both need the desired controller and the signal width as input values. The difference is that one works with the values from 0 to 255 and the other takes a percentage value between 0 and 100. The PWM controller can be selected with an enumeration, which is defined in the header file of the driver.

For a test of the PWM controller view here HowTo.

4.4.3.2 Motor Driver

With the PWM driver the PWM signals can be generated. The next step is a driver to initialize and control the motors and their controllers. Therefore the files "'b_motordriver.c"' and "'b_motordriver.h"' were implemented. It uses the PWM driver interface to set up the controllers.
To initialize all controllers as well as the motors the "'MotorDriver_init()"' method exist. It first sets the duty cycle to the highest value that still allows a frequency detection. After the return button on the keyboard of your computer was pressed the signal switches to the minimum value. The difference between both the highest and the lowest value defines the range a motor is able to use. Finally it needs another return to end the initialization. This procedure has to be done during every start up of the system, because the controllers are not able to store these values.
Apart from that the driver provides methods to set the speed of a single motor or all together. The motors can be selected with an Enum that is defined in the header file.

4.4.3.3 I2C Driver

The I2C Driver is the same we have programmed for the solar project, because we already tested it during that project. So we were sure our driver works and no changes had to be done. The files are named "'b_i2cdriver"'.

It provides various functions to transmit data with the I2C standard. To use this functions the controller has to be initialized once in the main program and before each transfer the method "'I2CDriver_open()"' has to be called with the needed speed setting of the used I2C standard. After that the transfer functions can be called with an I2C address and either a single value or multiple ones to be written or read. Finally the I2C controller has to be closed with the "'I2CDriver_close()"' method.

4.4.3.4 Accelerometer Driver

This driver provides interfaces to read the measured values of the ADXL345 accelerometer. For this the I2C driver is required to read or write into the registers of the sensor. Two files for the driver exist, the source file "'b_accelerometerdriver.c"' and the header file "'b_accelerometerdriver.h"'. The code is written on the basis of an old project.

For the initialization the driver sets up the I2C driver and starts the measurement mode of the accelerometer. No other initializations are made, maybe some better power control could be realized to safe some energy.

In addition to this three methods to get the measured values are provided. They simply read the corresponding registers over I2C. For every axis two registers provide the value, so they have to be connected and then can be returned.

For the test documentation take a look in tests.

4.4.3.5 Gyroscope Driver

Also a driver for the ITG-3200 MEMS gyroscope is implemented and provides methods to get the measured values. The gyroscope driver is named "'b_gyroscopedriver"' and has a source and header file. It requires the I2C driver to access the sensor. As a basis of the code serves an old project.

During the initialization some settings are made for the gyroscope. This can be done by setting different values in the configuration registers of the sensor. The settings made by this driver are:

- internal sample rate to 1kHz
- sample rate divider to 10
- full scale range to +/- 2000 degree/sec
- digital low pass filter bandwidth to 188Hz
- PLL with X gyro reference

After this settings the sensor needs 80 ms to get ready for the first measurement.

For all three axis are methods available to get the single values or all three together at once. The values are 16-bit two's complement and have to be made up of two 8-bit registers.

The test-protocol for the gyroscope driver can be also found in tests.

4.4.3.6 Magnetometer Driver

This driver provides methods to use the HMC5883L magnetometer. It initializes the sensor and reads the measured values. The source file is named "'b_CompassDriver.c"' and the header file "'b_CompassDriver.h"'. Both can be found in the code folder within the driver subdirectory. For this driver a basis from an old project also existed.

First the initialization method should be called, so the measurements work well. Through the initialization the sensor is set to the following configuration:

- 8 samples averaged per measurement output

- Data output rate to 15 Hz
- Gain to 5
- continuous measurement mode

Finally it sleeps 100ms, so the sensor can adopt the settings.
A function was implemented to read the raw values of the measurement. It reads the 8-bit registers of every axis and combines them to 16-bit two's compliment numbers.
Then a function exists, which should get calibrated values after running the calibration function. Till now these functions doesn't work very well.

For a test documentation look in tests.

4.4.3.7 ARM Linux FIFO Driver

The Linux FIFO Device Driver is needed to access the FIFO for the MCAPI communication. It is available for the Nios II processor architecture. For the project it was necessary to port it to the ARM architecture. Direct access to the physical address space is not possible, because the ARM processor has a MMU. Therefore the driver has to map the physical address space into memory. After that it is possible to access the FIFO via pointer.

For a test documentation look in tests.

4.4.4 Other Components

All error codes returned by the software are gathered in a header file named b_errorcodes.h. Each error is defined according to the following pattern:

#define ERR_<COMPONENT>_<DESCRIPTION> 44

This has the advantage that the error codes can be changed centrally for the whole project by editing the mentioned file. This improves the maintainability of the software significantly.

4.5 Physical Model

4.5.1 Concept for the model

There are a wide range of different shapes available. Each of them with its own advantages. To give an overview we will only introduce the most common ones.

- 2 rotors:
  They are called twincopter, bicopter or dualcopter and use only 2 rotors that are either placed beneath or stacked on top of each other. These are not as easy to control as others and can not carry much weight. Also the failure of one motor will result in a crash.
- 3 rotors:
  Tricopters have their motors either ordered in T or Y shape. To turn the model a servo is used to tilt the motor in the back. The weight they are able to carry is also not very high, but they are very agile. The failure of a motor also results in a crash.
- 4 rotors:
  This is the most common shape, because the construction is more simple to realize and they can carry more weight in form of batteries or smaller cameras.

- 6 / 8 rotors:
  These are mostly used to carry larger weights instead of doing tricks. When size is not a problem the motors are placed in a circle shape. This makes programming the steering algorithm easier. These constructions keep flying even when one or two motors fail. This makes them ideal to carry expensive photographer equipment or other things alike.

After our market research and weight calculation we settled on a six rotor design. This will allow us to carry about 1.5 kg of additional weight, which is more than our customer requested.

4.5.2 Decision of the material

Multicopters can be build from a wide range of materials with each of them having different advantages. In general it should be as light as possible but at the same time not easily breakable and not too expensive.

- Aluminium:
  Aluminum has a density of 2.70 g/cm^3 and is available in nearly every shape. The cost is acceptable and the material can be easily worked with. A 1m^2 x 3mm plate costs between 150€ and 230€ depending on the exact type chosen. The weight for that would be about 8.2 kg.

- Fibreglass(GFK):
  Fiberglass reinforced plastic has a density of 2.0 g/cm^3. The cost is lower than aluminum and the cost of a 1m^2 x 3mm plate is between 70€ and 120€ depending on the color. The total weight of it would be 6.0 kg.

- CFK:
  Carbon fiber reinforced plastic has a density of 1.5 g/cm^3 which is really good compared to GFK and aluminum. Unfortunately the cost is also higher. The same plate would cost us about 570€. Total weight would be 4.5 kg.

It would be ideal if all parts could be manufactured out of carbon fiber reinforced plastic, but the material is rather expensive. That quickly lead us to use fiberglass reinforced plastic instead. The material is just slightly heavier but therefore much cheaper.

We then started searching for a manufacturer that was able to produce all the required parts. After several months we still had not found one. All the offers were either to expensive or the precision was not high enough. This lead to the idea that we could use the milling machine, normally used to produce circuit boards, available in house. The disadvantage here is that we can not use every material and the size of the parts is limited to only 300 x 400 x 1.5 mm.

This lead to several design changes described in the next section and a material change. We are now using FR4. This material is also used for circuit boards and has a density of 1.85 g/cm^3, which is even better than the fiberglass reinforced plastic.

4.5.3 Construction

In our case we have multiple different requirements. Most importantly we have to carry 2 Kinect cameras and also a second computer for heavier computing tasks. Additionally the whole system needs to be as flexible as possible so that the weight can be freely shifted between front and back. This will result in a very good balanced model and provides the ability to expand or replace different components later on.

In addition our customer said it would be nice if the whole model could fit through a standard door. This added a lot more complexity. As explained earlier, all parts are normally placed in the center with the motors in a circle around it. This provides a perfect balance and is easy to build.

Unfortunately a standard door is only 90cm wide and with the needed 33cm rotors, a circle shape is not possible. This quickly lead to the H shape and also a few new challenges.

1. 1. Where can we attach all the required hardware?

2. 2. How and where can we mount the Kinect cameras?

3. 3. How to keep the balance throughout the model?

For point one we have come up with the idea to stack our hardware on top of each other into the free space between the rotors as seen in illustration 4.14 and 4.15.

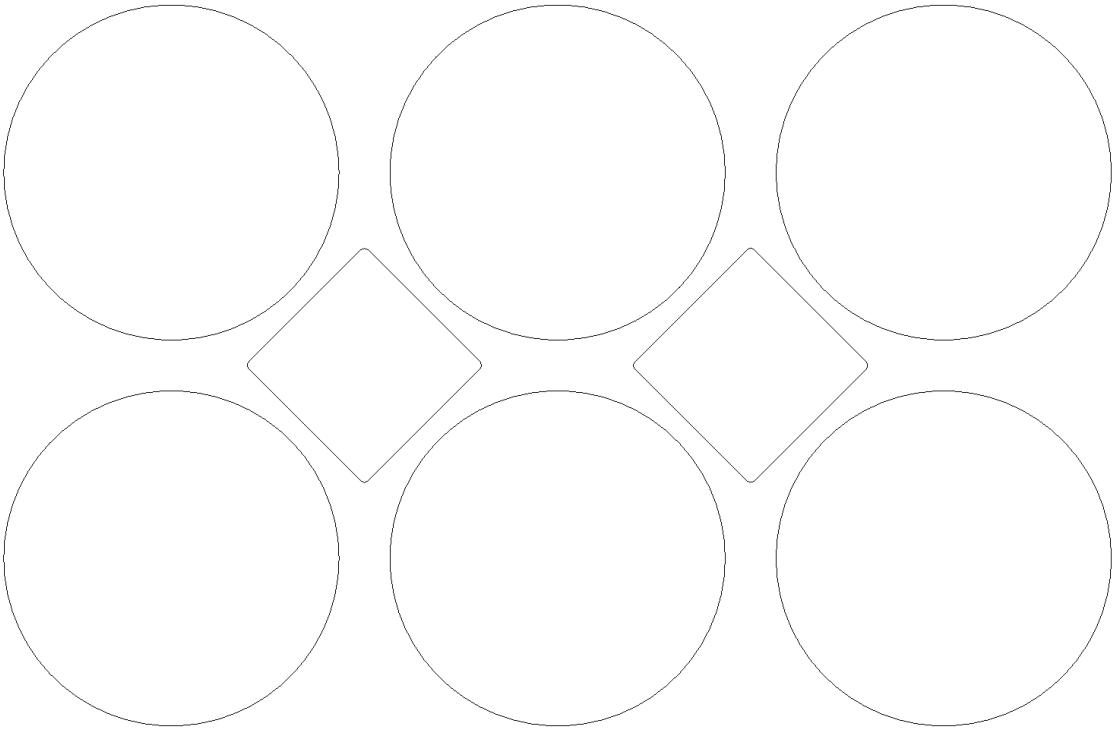


Figure 4.14: Frame schematic

This way the mass is distributed mainly on two points along the roll-axis. So to also reach a good balance on the cross-axis we need to be able to shift parts between these two points. To be able to do that, we use stackable plates to mount the required hardware.

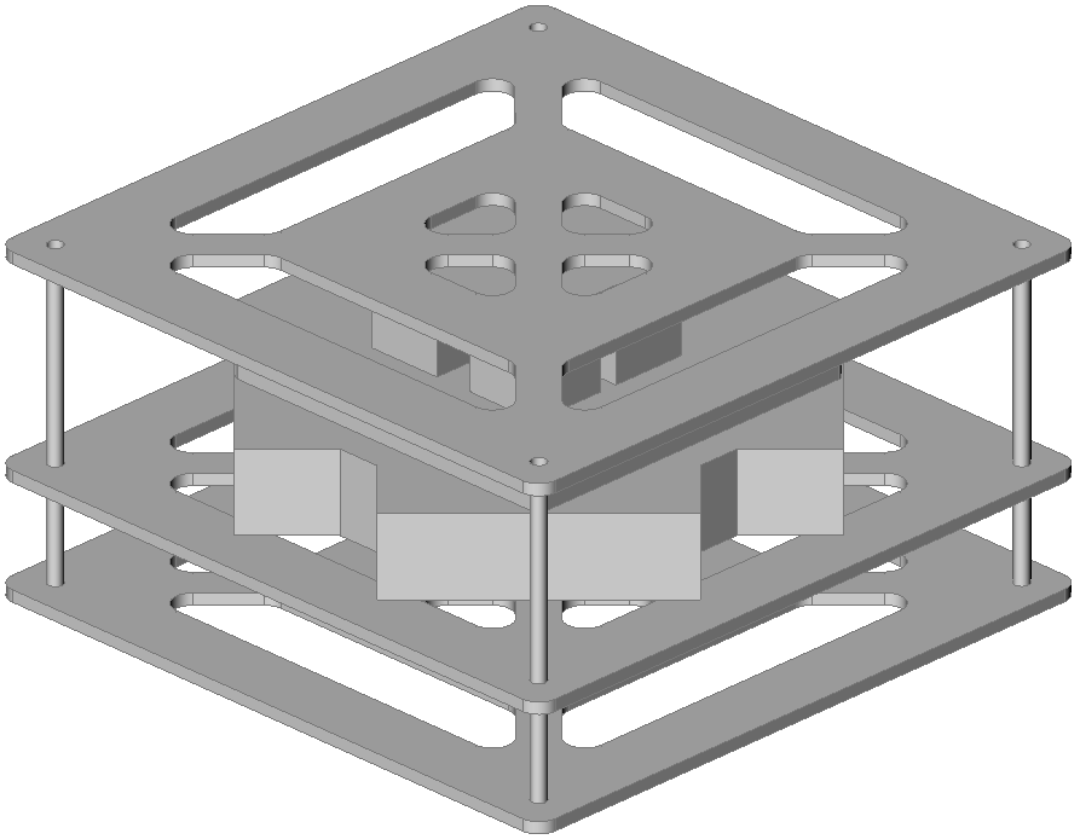


Figure 4.15: Example layout of a stackable construction

For the second point we developed a custom mounting solution that allows us to adjust each camera separately, which is shown in illustration 4.16. This bracket is mounted at the bottom and can be freely moved between front and back. This also helps to keep a well balanced model.

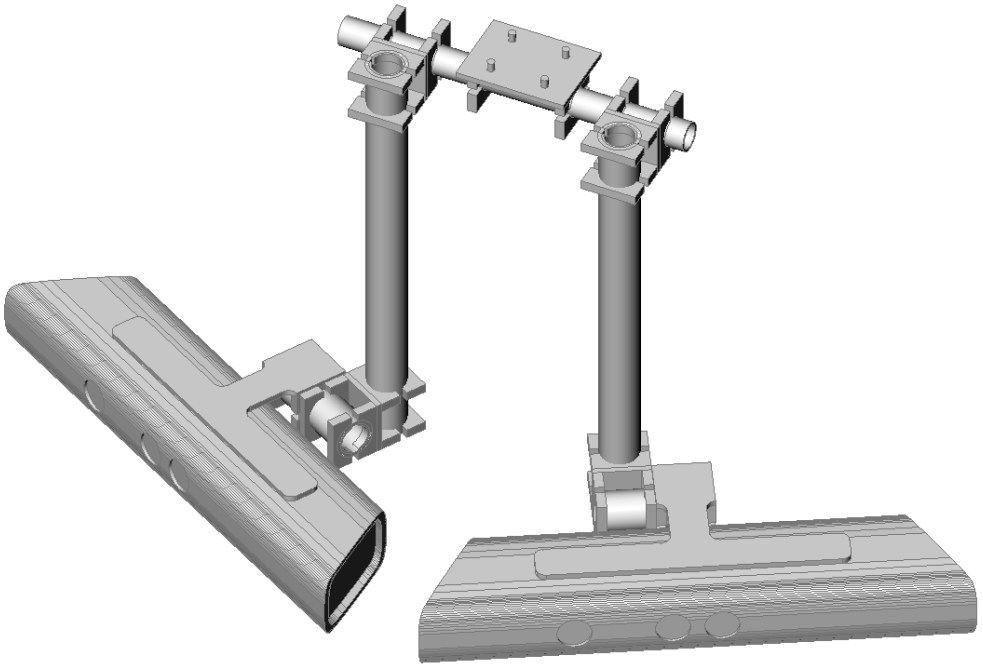


Figure 4.16: Camera holder schematic

To protect the rotors and prevent a crash during bumps we use a mold and a milling cutter to produce bumpers (4.17) out of hard foam. These are light and cheap and can be easily replaced if damaged. They are also sturdy enough to eventually carry smaller sensors later on.

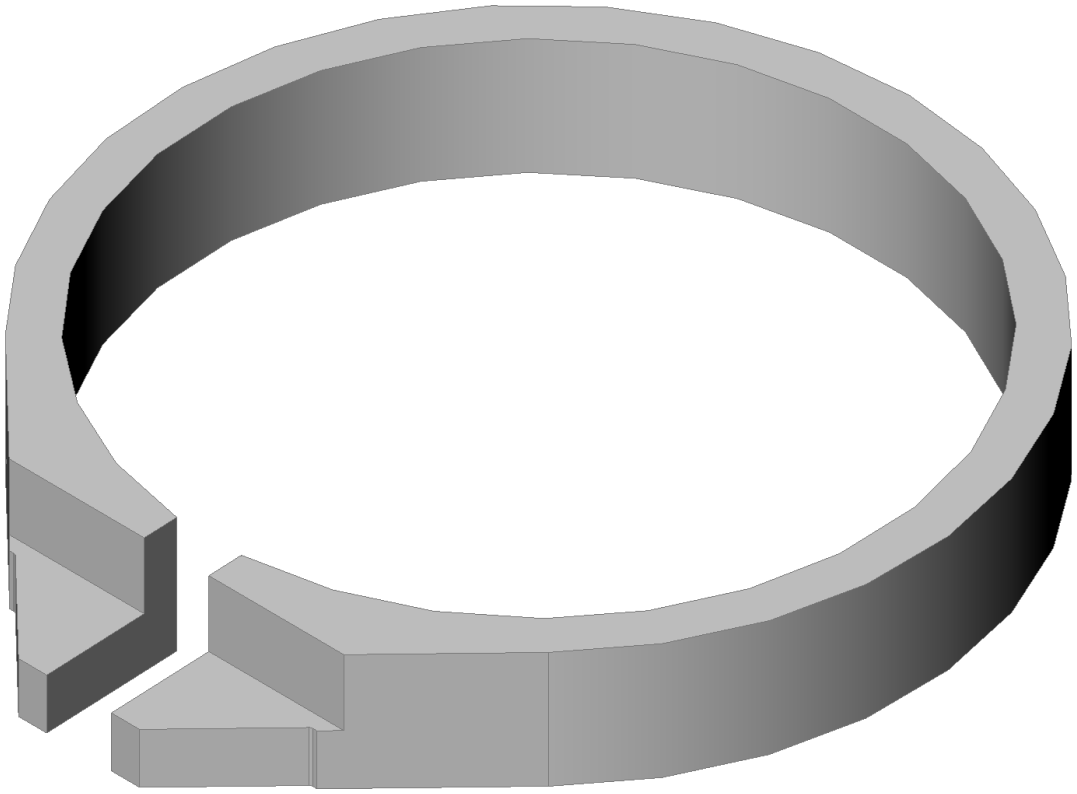


Figure 4.17: Bumper schematic

As explained in the material section above, we had several limitations in size and thickness due to using our in house milling machine. Therefore we had to modify our initial construction shown in illustration 4.18.

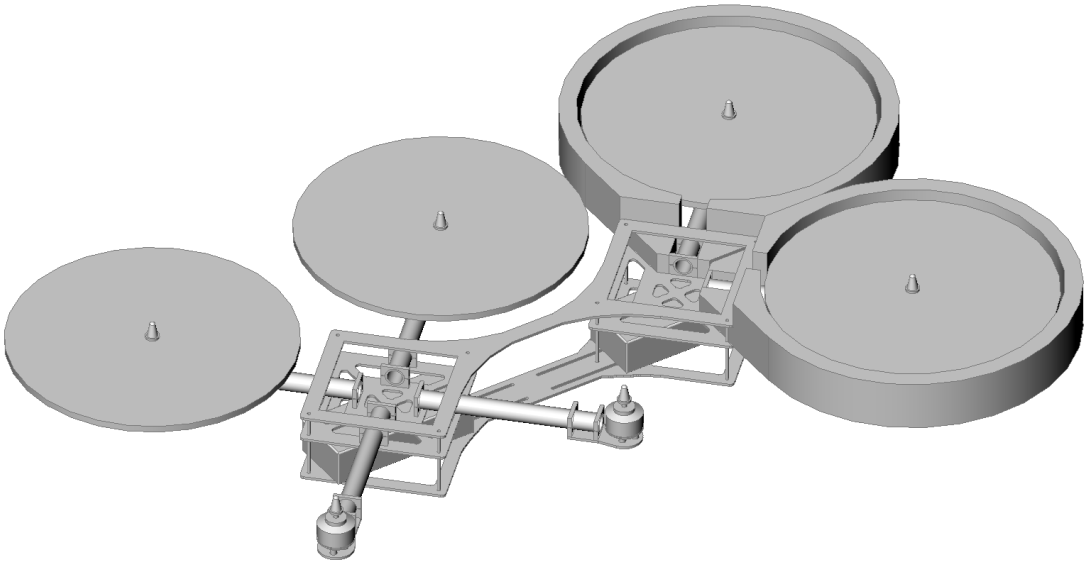


Figure 4.18: Model schematic

Both the upper and the lower connecting parts were to long to fit the machine. Also the required thickness of 3mm was too much. At that time we already had a few of the

stackable parts (illustration 4.19) manufactured in the new material to see if it holds up to the task. This lead us to the conclusion that we also had to reinforce some of the parts in use.

For the middle parts we filled the outer holes to reduce the flexing between the motor mounting points and removed the crossing in the center. This way we still enable a proper wire management.

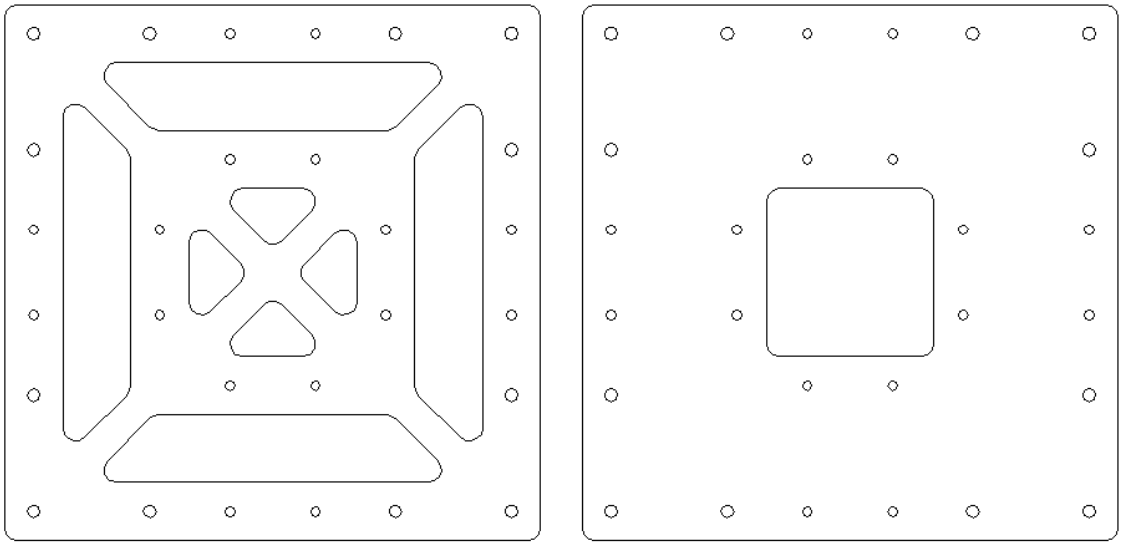


Figure 4.19: Old (l) and new (r) middle section

For the upper and lower parts, shown in illustration 4.20 and 4.21, that connect both towers the only solution was to split them up in two parts and provide nuts for proper fitting. Since we had to glue two layers together anyways to reach the required thickness of 3mm this was acceptable. In order to though maintain a high enough stability the cut is off center and both layers must be glued together with one of them shifted 180 degree.

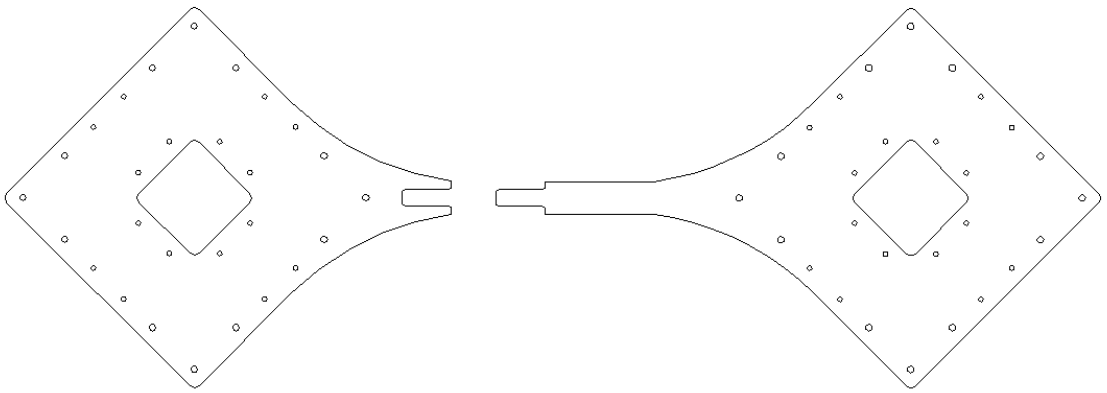


Figure 4.20: Upper section

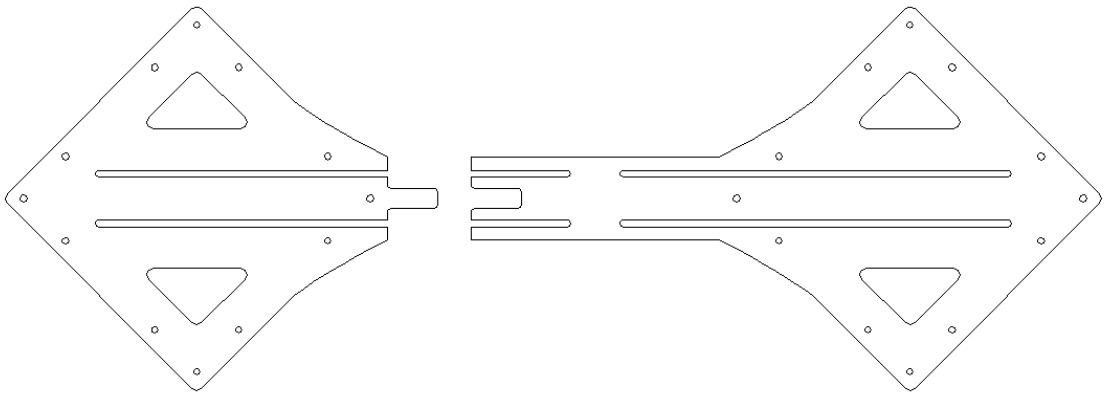


Figure 4.21: Lower section

All the other parts could be manufactured as planned before. See the table below for a complete listing of all FR4 parts.

- upper_part_1
- upper_part_2
- middle_part
- lower_part_1
- lower_part_2
- motor_mount
- kinect_mount
- connection_1
- connection_2

In illustration 4.22 you can see the newest 3D model

If additional parts are needed, please continue reading in the HowTo section.

4.6 Power Supply and Interface Circuits

4.6.1 Power Supply System

For the definition of the upper bound of the supply-voltage, we have to take a look on the accumulator system. In model-building LiPo- (Lithium-Polymer) accumulators are very popular. A LiPo-accumulator is composed of some single LiPo-cells (at least one). Each cell is able to provide between 3.5V and 4.3V supply-voltage. The accurate value depends on the current charge. The nominal voltage is defined as 3.7V.

A single cell is not able to provide an endless high current, in fact there is a value defined which specifies the nominal current-output of one cell as a function of its capacity. This value is known as the C-rate.

The nominal output can be calculated as:

$I\_out = \frac{charge[mAh]}{1000} * C$

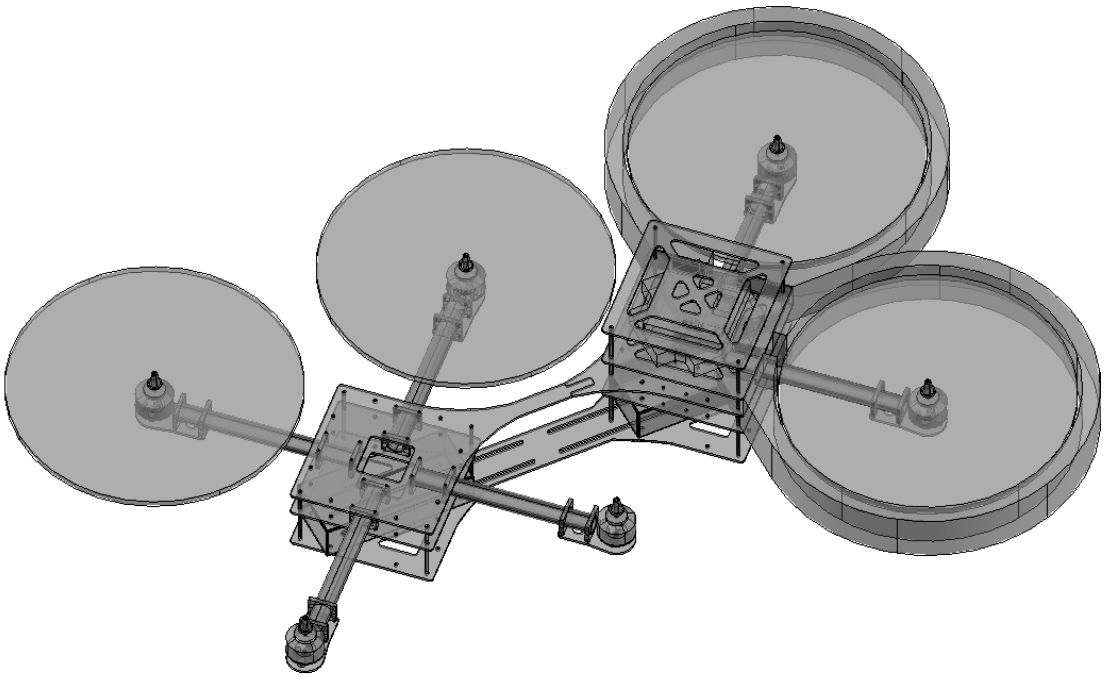


Figure 4.22: Latest version of the 3D model

For example a cell with a charge of 3000mAh and a C-rate of 15 is able to provide 45A constantly. Many cells are able to provide some extra current for a defined time (also known as maximum burst discharge). In this case the specification for the C-rate looks similar to 15C/25C.

As already said, you can combine some cells to an accumulator-pack. In general there are two different ways of combination possible:

- series connection (S-value):
  Cells in series increase the nominal supply-voltage by their quantity. For example: 2 cells in series produce 2*3.7 V = 7.4 V. In series-connection you can just use cells with the same capacity. If you violate that, weaker cells will be destroyed.
- Parallel connection (P-value):
  Parallel cells in an accumulator-pack increase the collective-capacity of the pack. For example a pack by 3 single cells with a capacity of 500 mAh makes 1500 mAh collective-capacity. Parallel cells do not increase the voltage supply. In this case it is very important to use similar cells (with the same C-rate) for building an accumulator-pack, otherwise some cells might be depth discharged and expand or, in the worst case, explode.

In Illustration 4.22 you can see an example for an 4S2P accumulator. You can also see the balancer-connection. The balancer is responsible for adapting the voltage in all cells. This is very important for preventing problems with unequally charged cells. When cells are unbalanced loaded, the weaker cells can fall in depth discharge while the accumulator-pack is in use or in over-charge when the accumulator-pack is in load. Depth discharge and over-charge can reduce the capacity significantly or destroy the cell completely. To prevent the depth discharge it would be wise to use an alert. Fortunately the discharge difference in one discharge cycle is not that big and we do not need an additional balancer on the model. It is sufficient that the accumulator is balanced during every the loading cycle. For the loading-procedure it is important to pay attention to the loading C-rate.

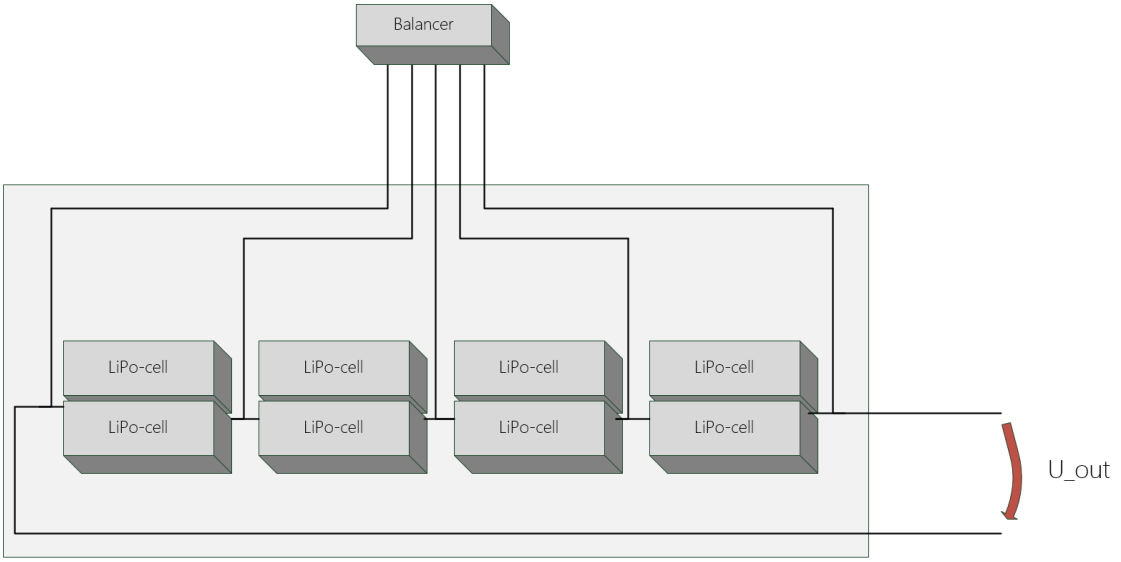


Figure 4.23: LiPo schematic

This rate specifies the maximal loading-current and must not be passed, otherwise the accumulator-pack will be destroyed. Furthermore it is better for lifetime to load with a little lower current than the C-rate specifies.

All LiPo-accumulators are sensitive in electrical, thermal, magnetic and normally in mechanical (there are some special versions in a hard case available) terms. The usable charge extremely decreases, when the temperature borders are violated. Mechanical deformation can destroy single cells in the accumulator and in the worst case the accumulator will inflame.

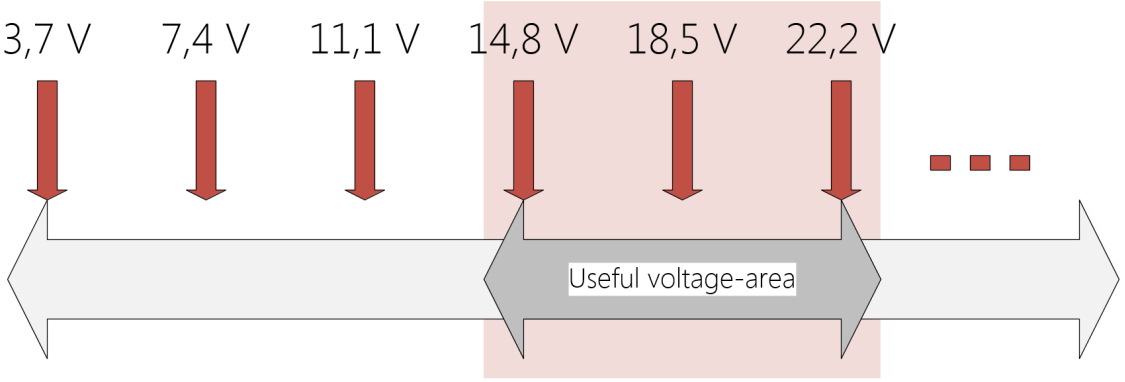


Figure 4.24: Voltage range

Illustartion 4.23 shows the voltage range. With the low-voltage-bound of 12V the smallest possible configuration of an accumulator-pack is 4S (14,8V). The upper bound is defined by the maximum value of series-connected cells, in modelbuilding normally 6S. So we defined a supply-voltage range between 14,8V and 22,2V (4S and 6S).

The Lower Bound of the overall C-rate of one accumulator pack is defined by the maximum consumed current (additional with some tolerance). The condition in the following

formula has to be satified to define the lowest C-Rate.

$C\_rate \geq n\_motor * I\_max * 1.2$

In illustration 4.24 you can see a diagram which shows the usable accumulators for a maximum consumption of 28.3 A. Accumulators left sided to the blue area have a too low capacity and don't seem to be reasonable. On the other side, accumulators on the right side are too big (the dimensions can not be fitted on the model). Furthermore the balance and the stability in static flight must be considered. So it is useful to separate the accumulator system into two or more parts.

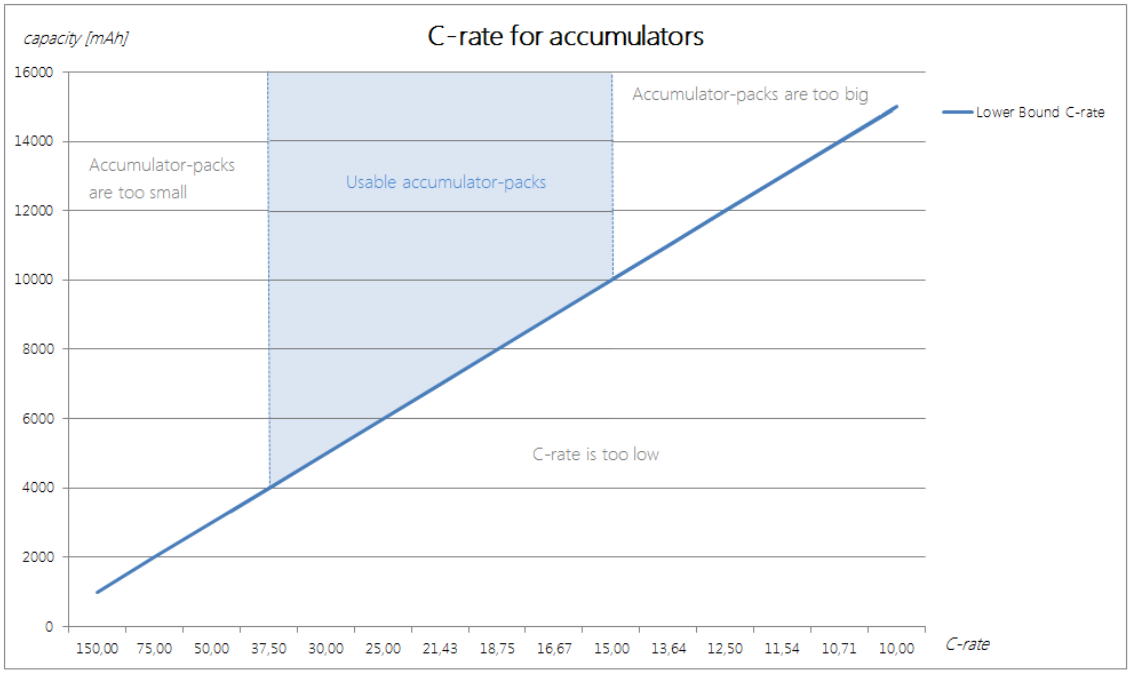


Figure 4.25: C-rate calculation

Due to some reasons we decide to use a 6S system (22.2 V):

- most of the motors for our purpose require a voltage of 22.2V
- accumulators with less cells have a lower capacity limit, so we would need more accumulators. The bad result of this is that there is more mechanical overhead of accumulators when you have to use more than one 4S accumulator to get the same capacity given by one 6S accumulator. More mechanical overhead results in more weight.
- Electrical power is the product of current and voltage. If you want to generate the same power with a 4S system and a 6S system, the 4S system needs an higher current. The problem at this point is that a higher currrent results in a higher loss of power and requires a wiring with a higher cross section (this is the same reason why power lines use an enormous high voltage). So we should use the highest usable voltage level to reduce loss and enable a small wiring.
- Accumulators with more than 6 cells and motors which can use a higher voltage level than 22.2V are oversized for our platform.
- The turnrate (the KV value in the motor specification) depends on the used voltage. A KV value of 630 means that the motor turns 630 times per minute with 1V. Now

with a higher voltage level, the motor is able to provide a higher turnrate. Because of the higher turnrate we can use smaller propellers to generate the same boost as there are used a smaller turnrate with bigger propellers. This is a very important point because we have to meet reqirements for an indoor flight.

The expected power of the model can just be estimated by means of the given values for the motors' power and the resulting boost, generated with the propellers. The result of that is just a momentary value.
The flighttime depends on how long the accumulator system is able to provide the energy which consumes the motors to generate these power.
To maximize the efficieny of the motors they should just need at most the half of their maximum consumed power to hold the model in a stable flight. It is necessary to have capacity upwards due to increase or decrease the motors' power to do flight movements. If the motors have to generate more power just for a stable flight, they have a loss of efficiency. This mean that they need proportional more energy than they generate more boost power. As a result the payload of accumulator capacity has a limit where more payload causes in a decrease of the maximum flight time.
Furthermore the flight behaviour has influence on the flight time because there is more energy for a more sportive flight behaviour necessary. But the intended use of the model promises a more static behaviour, so it seems useful to calculate with 25% additional power consumption.
The graph in illustration 4.19 is generated with a flighttime calculator written by a member of a mikrokopter forum[REC15]. This software may produce a little incorrect results but it provides a first direction where we have to look at.
The program calculates with gramm per mAh, given by our accumulator which has

$\frac{1377}{10000} = 0.1377 g/mAh$

The generated graph distinguishes between different waste ratings, depending on the flight modes hover, normal and sportive. As you can see in illustration 4.25 there is a flighttime of nearly 18 minutes with a capacity of 20000 mAh possible (in normal flight mode). The highest useful accumulator capacity is about 43000 mAh. More capacity results in a lesser flighttime because the motors have no linear relation between consumed power and outputed boost. Furthermore each motor has a limit for the maximum power. If the model is heavier than this limit, the model will not even make a wince. This fact is not visible in the graph due to some failings in the calculation program. The increasing graph at about 50000mAh depends on this problem too. Normally the graph will constantly decrease above the limit of 43000 mAh. So the limit of capacity will be about 40000mAh.

For testing purposes we just ordered a 6S accumulator with a capacity of 5000mAh[SLS15]. This capacity is high enough for do some measurements and initial tests but later on it is necessary to use bigger accumulators. The final configuration of the accumulator system has to comply some requirements:

- There should be an adequate flight time at least about 15 minutes. Therefore the accumulators system has to provide enough capacity. As shown in the graph above this will be about 20000mAh.
- The accumulator system has to handle with a high load effortless. Therefore each accumulator used in the system needs an adequate C-rating. It seems useful to calculate with a maximal consumed current of about 200A.
- The accumulator system should be mechanical distributed for a better weight balance of the model. It seems useful to use at least two accumulators, one in each stack

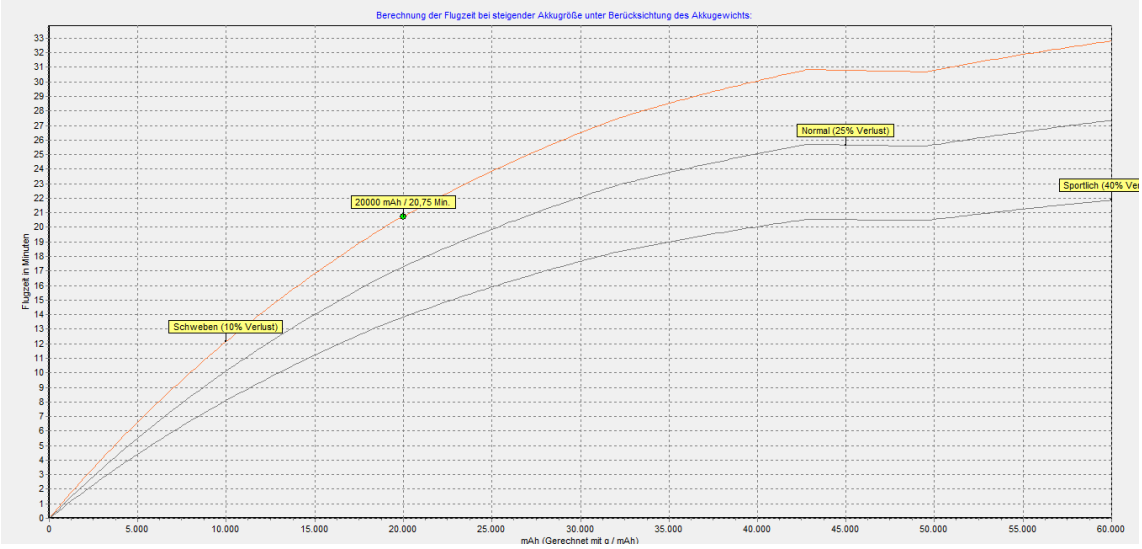


Figure 4.26: Estimated flighttime, depending on the accumulator capacity

of the model. For balancing the model it would be nice if the accumulators can be moved easily.

- Because of the distributed system there is a distribution platform necessary.
- The accumulators should be easily removable because they should not be loaded within the model.

For a further work on this project we recommend to use the SLS APL 10000mAh accumulator with a C-rate of 25C/40C twice[SLS15]. This accumulators have a high capacity an can be placed within in the stacks i the model.

4.6.2 Supply of the Circuit Board

The battery provides a voltage between 19V and 26V, but the DE1-SoC board expects a value of 12V and 3.5V. Therefore it is necessary to implement a power supply system on the extension board. We decided us to use the same controller that Frank Seifert used in his bachelor thesis, the LM5085[DAT15]. It worked with an input voltage between 12.8V and 17V and therefore it was impossible to reuse the circuit one-to-one. To dimension the different components correctly we used the online tool from Texas Instruments. But after assembling the circuit, the system did not work as expected. Although the input value was between 19V and 26V the supplied voltage was not 12V like expected, but 1.25V.

The following chapter describes the circuit we used and additionally describes some assumptions for the possible error source and which ones could be excluded already.

Description of the Circuit

Illustration 4.26 shows the circuit diagram of the power supply system, which was generated with the online tool from Texas Instruments.

To understand the wiring it is useful to have a look on the different pins of the controller LM5085.

ADJ - Current Limit Adjuct

It is the first pin and it is connected above a parallel connection of the capacitor Cadj and the resistor Radj. The capacitor should have a value of 1000pF. It is used to filter

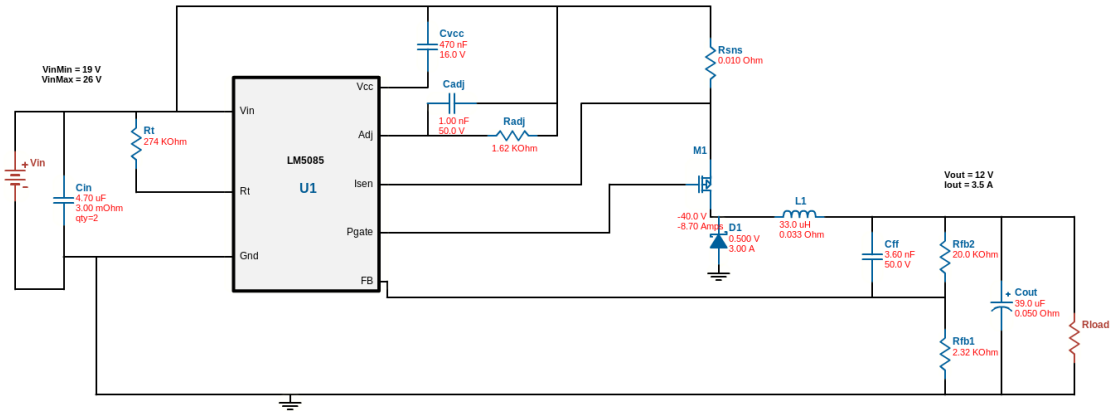


Figure 4.27: Circuit diagram of the power supply system

noises and it also prohibits unwanted switching of the current limit comparator due to input voltage transients.

RT - On-time Control and Shutdown

RT is the second electrical contact of the controller and is connected with the input voltage over a resistor Rt. It is responsible for a continuous mode switching frequency. If the contact is connected with ground, the controller will shutdown.

FB - Voltage Feedback from the Regulated Outpout

The third pin named FB serves for the feedback control. The regulation level is 1.25 volt. With an input voltage between 19 and 26 volt it was possible to measure a relatively steady value on this pin between 1.24 and 1.27 volt. The controller defines his off time out of this input value and also the input value from V_{in}.

GND - Circuit Ground

Pin number 4 connects ground with the internal circuit ground.

ISEN - Current Sense Input of Current Limit Detection

ISEN is the fifth pin and must be connected with the input voltage via a sense resistor Rsns. The resistor is used for the over-current detection. The measurement doesn't reveal any difference between the input voltage and the input value of the pin.

PGATE - Gate Driver Output

The sixth pin of the controller should be connected with the gate of the MOSFET M1. The output value fluctuates between Vin when the MOSFET is off, and the value of the seventh pin VCC when the MOSFET is on. The rise and fall times depend on the MOSFET gate capacitance and the source and sink currents provided by the internal gate driver. It is important that the on-time at this pin is not lesser than 150ns.

VCC - Output on the Gate Driver Bias Regulator

The seventh pin of the controller is the output of the negative voltage regulator and must be connected to the bias of the MOSFET over a series connection of Rsens and Cvcc. Cvcc has to be a 470nF capacitor and provides the high surge current for the MOSFET gate at each turn on. It also provides noise filtering and stability for the VCC regulator.

VIN - Input Supply Voltage

The last pin is connected with the input voltage directly. This input refers to the supplied voltage by the accumulators. Depending on the state of charge the supplied voltage can vary between 19V and 26V.

EP - Exposed Pad

The exposed pad is fixed at the bottom of the component. It must be connected to ground. The surface of the ground connection has to be as big as possible to help with heat dissipation.

Voltage Divider

RFB2 and RFB2 constitute a potential divider.

$V_{out} = \frac{1.25V*(RFB1*RFB2)}{RFB2}$

with $V_{out} = 12V$ results:

$9.6 = \frac{(RFB1*RFB2)}{RFB2}$

It should be dimensioned between 1 and 20k Ohm.
RFB1 = 2.32k Ohm
RFB2 = 20k Ohm

$\frac{(RFB1*RFB2)}{RFB2} = \frac{29kOhm*2.32kOhm}{2.32kOhm} = 9.62$

Root Cause Analysis

After extensive inspections the usage of wrong components could be excluded. Also a connection between the second pin and ground of the controller, which would result in a shutdown of the controller, could be disqualified.
Tests with 20 ohm resistance and applied oscilloscope has shown that the controller regulates. Unfortunately to 1.25V instead of 12V.
It was not noticed, that the connection between the exposed pad and ground was absent. Therefore the refrigeration of the controller aborted. There is a high probability that the inner circuit of the controller was defective.
An other possibility is that the MOSFET got damaged because of its high sensitivity.

Current Status

At the beginning two boards were made. Both of them are currently useless concerning the power supply. Through tests the wiring was damaged. For board tests it is possible to use a power adapter.

The circuit should be build up on an experimental board and tested to find out if the missing connection between the exposed pad and ground or the MOSFET was the reason for the misbehavior.

5. Project Documentation

5.1 Project Management Method

In the project we applied Scrum which is a project management framework whose most important aspects will be described in this section. It is designed for teams of 3 to 9 persons.
The motivation for Scrum is that in many projects the requirements are not clear at all at the beginning of the project. The project often is to complex and fails in the very end for example when all components shall work together for the first time.
To avoid this Scrum implements an iterative and incremental approach. In Scrum the team maintains a so called backlog in which it keeps userstories that represent the needs of the customer. These userstories are sorted by their priority.
The project phases in Scrum are called sprints and last about 4 weeks. Before one sprint the team plans the sprint by deciding which userstories can be realized in the next sprint and how many Storypoints are assigned to a specific story. These Storypoints are the time units used in Scrum. Also the Userstories which are small enough to complete them in one sprint have to be divided in multiple tasks that should be completable in one day. The selected userstories are passed to a sprint backlog.

Within the sprint the team works on the tasks in order to complete the userstories. Each day there is a so called Daily Scrum which should be no longer than 15 minutes in which the team meets and tries to inform everyone about the current status.

After the sprint the result should be a runnable version of the product that is being developed. There should also be a retrospective in which the team has a look on its former work method to improve it and work more efficient.

Another important point in Scrum is that there are 3 roles in Scrum that have to be assigned. There is a Product Owner who is responsible for the economic success and the features of the product. Therefore he regularly has to keep contact with stakeholders e.g. the customer that wants to buy the product.
The second role is the Scrum Master. He doesn't belong to the team that develops the project but works together with the team in order to support the success of the Scrum method. Therefore he for example organises meetings and tries to eliminate disturbances that detain the team from efficient working.

The third role is the development team which has to realize the features that are mentioned in the sprint backlog. It also has to help in the planning of the sprint for example by guessing the time a feature in the product backlog needs to be implemented.

5.2 Realization in the Project

In the project we didn't have such a strict role distinction. The roles we assigned were:

- Product Owner: Simon Krais
- Scrum Master: Daniel Klitzke
- Development Team: Yvette Groner, Hannes Heinisch, Daniel Klitzke, Simon Krais, Christoph Loeser, Christian Mueller

Through limited time resources it was not possible to keep the Scrum Master out of the development team.

One sprint of the project had a duration of 4 weeks. Before the sprint we everytime arranged a customer meeting to talk about the results of the foregone sprint and to determine which features are most important for the next sprint. There also was a retrospective of only the project team as well as a retrospective where problems and successes in the process had to be presented to other teams. During the sprint we had one day a week to work on the project. Mostly in the morning of this day the Daily Scrum was held.

5.3 Tools

There were several tools used to help in the project coordination process. The main task management system was OpenProject from http://www.openproject.org. This is a web application that was installed on a server accesible from the internet, in order to manage the userstories and tasks, log progress, and so on. Another tool that helped us to synchronize our files, especially the documents we made for documentation, was a cloud storage service called Seafile that was also installed on an own server.
When it comes to software development we had the need for a good revision control system. Therefore we chose Git which was also installed on the server. It helped us to keep track of the different versions of our code. Furthermore it also manages to solve conflicts and prevent data loss, which could easily happen if multiple people work on one file.

5.4 Sprint 1

The first sprint of the autonomous multicopter project consisted mainly of setting up the development environment and doing research on the topic.

Furthermore the project was split into three categories: the hardware, the extension board and the SoPC.

On the hardware side it was planned to develop a first 3D model of the frame and the brackets to mount the Kinect cameras. Also, calculations of the possible flight time and payload were planned to be done, to find out which components would be necessary.

For the extension board, the given circuit board from a bachelor thesis has to be inquired to figure out which interfaces are already available and can be tested. Also, all additional interfaces should be defined and a first board layout should be done.

On the SoPC side, two NIOS-II soft processors have to be synthesized into the FPGA and connected among themselves and with the HPS via FIFO bridges. To test the communication over the FIFO bridge, test programs have to be written and also a communication concept has to be developed. In addition, first interface components should also be synthesized. If possible, a first MCAPI program should also be written.

The results on the hardware side were a summary of possible parts and a first 3D model of the frame and the brackets for the Kinect cameras. Also the calculations of the flight time and the rough payload have been done.

On the given circuit board the reusable parts were found. The necessary interfaces were defined and it also was decided to use the DE1-SoC as main board to reduce the effort of developing a new extension board.

In the SoPC a first test system was established with two NIOS-II processors and all FIFO bridges between both NIOS-II processors and the HPS. To test the communication between all processors over the FIFO bridges, test programs which directly access the bridge registers were written and the communication test went off positive. Because both NIOS-II use the same On-Chip memory, a test program was also written to find out(,) if there are any interferences. The result of the test was that no interferences exist. Furthermore a first test program using the MCAPI was written, but could not be tested.

During the sprint, the tools on the VM crashed several times which causeda lot of problems and delays in the development of the SoPC and test programs. Also, the defined user stories were to imprecise which caused irritation what has to be done exactly.

5.5 Sprint 2

In the second sprint first orders were made and the main development began.

It was planned to order first parts of the required components for the multicopter to check if the calculations are correct. Also the 3D model has to be revised and a landing gear should be developed. To manufacture the frame and the brackets, a possible manufacturer has to be found. If possible the first parts of the multicopter should be assembled.

The extension board has to be designed and a board manufacturer has to be found to produce it. Also the existing USB controller has to be tested.

On SoPC side, the version of the NIOS-II has to be changed to the more efficient version. Also the Linux FIFO driver was planned to be ported from NIOS-II to ARM architecture. Therefore information about the ARM interrupt system has to be gathered. To decide which HPS-to-FPGA bridge should be used and developing a communication concept is also planned in the second sprint.

At the end of the sprint, the 3D model was revised and production ready. Also all parts were ordered and motors and ESCs were evaluated. The USB controller was tested and the power supply system was designed. Additional the version of the NIOS-II was changed. The FIFO driver was successfully ported to the ARM processor and a document about the ARM interrupt system was created. The decision which HPS-to-FPGA bridge should be used felt to the lightweight bridge. Also a first communication concept was developed.

The first MCAPI tests were not possible because of the lack of memory. This is ag-

gravated by the fact that the SoC board changed from the SoCkit board to the SoC-DE1 board. Also delays in delivery of components for the extension board caused some problems. Furthermore some problems with finding a manufacturer for the frame production occurred.

5.6 Sprint 3

For the third sprint it was planned to create a milling fixture and produce external frames. Also the first model parts should be combined and an initial function test was planned to be done.

For the USB controller a circuit board should be drawn in Eagle and also the layout of the extension board itself. Also A manufacturer for the boards has to be found.

The SoPC has to be redesigned, especially the memory distribution. The communication concept has to be implemented with the MCAPI. Furthermore the necessary interfaces should be synthesized in the FPGA and matching device drivers for these IP-Cores.

In addition the current report has to be converted to LaTeX.

As a result of the sprint, the necessary IP-Cores were found with all required drivers. The extension board and the USB board layout were finished. Additionally the required components needed and the board itself were ordered. Also a first software architecture was made while the SoPC is still in work. The milling fixture were created and the external frames were produced.

During this sprint, problems with Quartus II 32bit version and the SoPC occurred and further problems with the new board. This problem could be solved by switching to the 64bit version of Quartus II.

5.7 Sprint 4

In the fourth sprint it was planned to fix the SoPC problems with the interrupt invoke on the ARM and modify the device drivers so that they work on the SoPC system.

Also the programming of the flight control should begin and the last parts for the frame should be manufactured. Another task in this sprint was to synthesize the IP-Cores for the I2C, SPI, UART, USB and PWM input/output into the FPGA and test the interfaces. The drivers also have to be modified in this sprint.

The results of the fourth sprint was a working MCAPI because the interrupt invoke problem on the ARM was solved. Also the I2C IP-Core was synthesized and the driver was tested.

Furthermore the PWM output IP-Core was integrated and the driver was modified to work with our system. Still to do are the tests and integration of the SPI, UART, USB, PWM input and the XBEE interface. Also the board was produced but the power supply was causing problems. The reason for this problem was not found yet.

5.8 Sprint 5

In the fifth sprint, it was planned to bring the project into a solid state for the next team which continues working on it. Therefore the documentation has to be completed and

HowTos to several components and the setup of the development environment have to be written. Also if possible the multicopter should be completely build up. If possible some errors in the software drivers should be fixed.

As the result of the sprint the model has been built up for the most part. Also last changes on the software drivers were made in order to establish proper functionality.
Another big part was the composition of the final project report. As a help for the further development of the project a set of HowTos was written in order to provide guides for the most important task that could appear when continuing the project.

6. Project Evaluation

This section offers a short retrospective of the project. It summarizes which parts could be realized and which could not be realized. Also suggestions which parts can be realized in future projects are made in the last sub-section.

6.1 What Could Be Realized

Most of the project goals could be realized. The physical model could be built up with some problems in the selection of possible materials by using circuit board material as an alternative. Thereby all components that are necessary for appropriate flight characteristics of the model like motors, motor controllers and accumulators were selected and mostly integrated into the model. All electronic components are powered through a power supply circuit that has been designed by the project team. It has been built up on a circuit board that was specially manufactured for our purposes. Also all necessary interface circuits for the sensors and actors have been built up on the same circuit board. Within the project also a SoPC could be realized that mainly consists of three processors. Two of the processors are NIOS-II soft processors while one of the processors is an ARM processor. All RAMs used by the processors are separated which prevents any interference between memory acesses. Also the communication between the processors through FIFO bridges could be realized. Another thing that has been mostly realized is the implementation of the needed hardware controllers, to interact with sensors and actors, in the SoPC. The realized interfaces encompass I2C, PIOs and some PWM outputs. Also the most important sensors like the gyroscope, the accelerometer and the compass could be successfully connected to the system.
Regarding the software components the drivers for the mentioned interfaces as well as drivers for the most important sensors have been realized. Some of these sensor drivers were not tested sufficiently. A Document about which of the sofware components have been tested and which require more work can be found in tests. The requested communication between the different processors through the MCAPI was realized and tested. To enable this also a linux driver was ported to the ARM architecture. A software architecture for the control system has been developed but couldn't be implemented through time reasons.

6.2 What Could Not Be Realized

During the project some major problems occured which made the realization of some of the planned features impossible. Regarding the physical model there were some big

problems with the realization of big parts of the frame. Those should first be realized in carbon fiber but the material costs were far to high. Also the use of glass-fibre reinforced plastic was not possible because the milling of the material would have been to expensive. As an alternative the parts had to be realized in FR4 which is a common material for manufacturing circuit boards. This solution has also a positive aspect because damaged parts can be easily replaced by rebuilding them from cheap circuit board material.
Another important point which couldn't be realized completely is the power supply. It is currently realized through a linear controller which is very inefficient. A solution calculated with a tool provided by Texas Instruments didn't work properly torough unknown reasons. There are also restrictions regarding to the interfaces. What couldn't be realized were two additional USB ports which should be added to the existing two USB ports on the DE1-SoC board. The USB controllers were purchased and also added to the circuit board with the interface circuits and the power supply. The problem here was that the controllers have to be connected over an ULPI interface but a fitting ULPI IP-Core was not available for an acceptable price. The connection to the DE1-SoC board could may be realized in a following project by purchasing or developing an appropriate IP-Core. Also the integration of interfaces like SPI and UART couldn't be realized through time reasons. This also could maybe realized in a following project and shouldn't take to much time.
As already mentioned in the previous section some software drivers for the sensors require more work and testing. A list of which drivers were tested successfully can be found in tests. Also the software architecture of the control system was designed but not implemented.

6.3 What Could Be Done in Further Projects

As explained in the previous section some of the project aims couldn't be reached through technical or time reasons. However the realization of most of them isn't impossible and can be done in future projects. What could be improved is summarized in the following section. Multiple How-tos that help to continue the work on this project can be found in HowTo As mentioned before FR4 was chosen as main material for the multicopter's frame. This material does meet our requirements but isn't the optimum choice. Maybe a better material as well as a fitting and cheap manufacturing method could be found in order to manufacture an adequate frame.

Regarding the circuit board that contains the power supply as well as some interface circuits also several optimizations could be made. The circuit board layout was planned in EAGLE and the board was manufactured according to this plan. Afterwards some changes had to made especially regarding to the power supply. A functioning but non efficient version of the power supply was soldered on an extra board. A more efficient version could be integrated into the circuit board layout in order to get a cleaner result.

There is also remaining work regarding the SoPC. Some interfaces couldn't be synthesized in the FPGA through time reasons. The concerned interfaces that could be easily integrated in the FPGA are SPI, UART and PIO inputs that can be used to read the outputs of the RC receiver. A tutorial how to integrate new interfaces into the FPGA can be found in HowTo. Also the two external USB interfaces could not be connected to the ARM processor. These interfaces need an ULPI controller that should be synthesized in the FPGA, but no fitting IP-Core that was available for acceptable costs was found. In a future project such an IP-Core could be realized in the FPGA either by searching for an appropriate ready to use solution or developing an IP-Core on its own.

Also the software aspect needs some further work. All software drivers to the used interfaces have been realized but not every sensor that is connected to them has a fitting

software driver that was tested appropriately. Also other software components like the flight control itself and some filters are not implemented till now but contained in a software architecture that was designed within the project. A summary of which software components could be realized and what was tested can be found in tests.

7. Lessons Learned

In the project there was success as well as failure. There are things that worked quite well and also things that could be improved in future projects we execute. These both aspects will be discussed in the following section.
Regarding to the planning phase of the project we have learned the following. First of all it is very important to keep contact with the customer. Often there are misunderstandings especially if the customer is familiar with the sector of his product but the developers are not.
The most important point here is that the team or especially the product owner should react very fast if something is unclear and talk to the customer again.
Another important thing we learned was that also regular meetings within the project are very important and in these meetings each person should explain on which part he or she is working. Also results of this work should be presented periodically. This gives each person a better overview of the project and adds more control to the development process. Otherwise there could be times where tasks get delayed because nobody cares for them anymore.
A point that belongs to this is that these results should also be documented instantly in order to be able to measure the project progress.
Also a probably used project management tool has to be updated regularly. Otherwise there is no sufficient overview over the project progress.
We also made the experience that a assumed little change in the system architecture during the development process can cause major problems that delay the development of the product significantly. In our case this was the change of the SoC Board which led to the need for another development system in terms of another Ubuntu installation with a 64 bit version of Quartus-II. Such decisions should be carefully considered.
Another thing that could be improved is that if there are parts that have to be manufactured by external companies the planning should contain more than one possibility. If one manufacturer can't realize the task to the right conditions another one could be taken as an alternative. During the project there was a serious delay in the manufacturing of the parts for the physical model because of an unreliable supplier.
Regarding to the presentations there are also some things that could have been better. First of all we should have scheduled some more time for the preperation of the presentations. Also the presentations should be structured in another way. The presentation should always start with an overview over the project or the system. No assumptions that the Audience remembers anything from foregone presentations should be made.

8. Conclusion

Although the system is not finished till now the main goals could be mostly met.
The main vision of the customer of a multi processor system with separated rams could be mostly realized. Also many ideas like the realization as a SoPC to keep the system modular were integrated into the project. Another important requirement that could be realized was to abstract the communications over FIFO bridges through a middleware called Multicore Communications API. Also a needed linux driver to bring this on the ARM architecture was developed within the project which can be used also in future projects.Unfortunately there are also parts which couldn't be realized within the given time.
Most of the software couldn't be realized through time reasons which was mostly caused by problems we experienced by using the Altera tools. We definitely think that the project offers a solid base for further development. The modular approach offers great possibilities to integrate new features and ideas into the project. We hope that the project experiences active development in the future to bring it into the air at some day.

Bibliography

[Ayd15] M. Aydin, "Brushless Permanent Magnet Servomotors," Jan. 2015.

[BE415] "BE4108 Datasheet," http://www.dys.hk/ProductShow.asp?ID=10, Jan. 2015.

[Bra06] T. Braeunl, *Embedded Robotics*, 2nd ed. Springer-Verlag Berlin Heidelberg, 2006.

[BST10] K. Berns, B. Schürmann, and M. Trapp, *Eingebettete Systeme*, 1st ed. Vieweg+Teubner Verlag, 2010.

[Bue] R. Buechi, "Brushless Motoren, Grundlagen der Technik," *RC Car Technik*.

[Chu12] P. P. Chu, *Embedded SoPC Design with NIOS II Processor and Verilog Examples*, 1st ed. John Wiley & Sons, Inc., Hoboken, New Jersey, 2012.

[DAT15] "LM5058 Datasheet," http://www.ti.com/lit/ds/symlink/lm5085.pdf, Jan. 2015.

[DE115] "DE1-SoC User Manual," http://www.terasic.com.tw/cgi-bin/page/archive_download.pl?Language=Taiwan&No=836&FID=fb723f964a1d785b94cd7880d8f Jan. 2015.

[EMA15] "Emax MT3515 Datasheet," http://www.emaxmodel.com/views.asp?hw_id=1297, Jan. 2015.

[ESC15] "BullTec ESC 40A set," http://www.premium-modellbau.de/Brushless-Regler-BULL-TEC-Brushless-Regler-6x-BULL-TEC-Multicopter-Brushless-Regler-40A–2S—6S-Lipo-Opto-SimonK-N-FET/a47806162_u2344_z1d77a1d9-12c2-4a53-9bf3-ff595e5c0ad1/, Jan. 2015.

[Hau13] M. Haun, *Handbuch Robotik*, 2nd ed. Springer-Verlag Berlin Heidelberg, 2013.

[HLN12] J. Hertzberg, K. Lingemann, and A. Nüchter, *Mobile Roboter*, 1st ed. Springer-Verlag Berlin Heidelberg, 2012.

[Moh14] B. Mohammad, *Embedded Memory Design for Multi-Core and Systems on Chip*, 1st ed. Springer Science+Business Media New York, 2014.

[OFF15] "Altera Offers Complete USB 2.0 Device Controller Solution," Jan. 2015.

[REC15] "Flugzeitrechner," http://svn.mikrokopter.de/mikrosvn/Projects/Flugzeitrechner/, Jan. 2015.

[Sch15] H. Schenk, "Der Standschub von Propellern und Rotoren," http://www.rc-network.de/magazin/artikel_02/art_02-0037/Standschub.pdf, Jan. 2015.

[SLS15] "SLS XTRON 6S 5000mAh Datasheet," http://www.stefansliposhop.de/liposhop/SLS-XTRON/SLS-XTRON-40C/SLS-XTRON-5000mAh-6S1P-22-2V-40C-80C::1072.html, Jan. 2015.

[TIG15] "T-Motor MT3515 Datasheet," http://www.rctigermotor.com/html/2013/Professional_0912/50.ht Jan. 2015.

[U315] "T-Motor U3 Datasheet," http://www.rctigermotor.com/html/2013/Power-Type_0928/90.html, Jan. 2015.

[USB15] "USB 2.0 Function Controller," http://www.altera.com/products/ip/iup/usb/m-sls-usb20.html, Jan. 2015.

[Wie12] J. Wietzke, *Embedded Technologies*, 1st ed. Springer-Verlag Berlin Heidelberg, 2012.

[XBE15] "Official Site XBee," http://www.digi.com/xbee/, Jan. 2015.