Project Quadrocopter

Automotive Systems Master

Software Based Automotive Systems

ASM2-SB / SS 2010

XBee Communication



*Prof. Dr. rer. nat. (Purdue Univ.) Dipl.-Ing. Jörg Friedrich*

*Tutor: Dionysios Satikidis*

Joachim Schwab

Sreekanth Sundaresh

Nina Vetlugina

Marc Weber

General document information

This document provides information about the communication between the quadrocopter and the base station. The transferred data provides information for visualizing the attitude and actual state of the system as well as the possibility to read out and change system parameter. XBee Pro RF modules are used to transfer the data wireless.

For system parameters and states, there are used different data types. They are listed in the table below. In the flight control no floating-point unit is used, therefore all types are integer based.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Name** | **Sign** | **Width [Bit]** | **Value range** | |
| bool | (unsigned) | 1 (8) | TRUE or | FALSE |
| uint8 | unsigned | 8 | 0 to | 255 |
| int8 | signed | 8 | -128 to | 127 |
| uint16 | unsigned | 16 | 0 to | 65535 |
| int16 | signed | 16 | -32768 to | 32767 |

Table : Data types, used in the flight control

Table of contents

1 Calibration parameter 1

2 System states 2

3 Communication / Messages 4

3.1 Parameter Request (base station 🡪 quadrocopter) 6

3.2 Parameter Response (quadrocopter 🡪 base station) 6

3.3 Parameter Update (base station 🡪 quadrocopter) 6

3.4 Status Request (base station 🡪 quadrocopter) 7

3.5 Status Response (quadrocopter 🡪 base station) 7

Figure index

Figure 1: General message layout 4

Table index

Table 1: Data types, used in the flight control I

Table 2: Calibration parameter 1

Table 3: System states 3

# 

# Calibration parameter

To be able to configure the quadrocopter’s flight control, the wireless connection over the XBee modules is used. The following table shows the parameters, which can be changed (the values of the PID controllers and the parameters for sensor filtering).

|  |  |  |  |
| --- | --- | --- | --- |
| **Parameter** | **Abbreviation** | **Data type** | **Range / Unit** |
| P-value for roll control | kpR | uint16 | [100/sec2] |
| I-value for roll control | kiR | uint16 | [100/sec3] |
| D-value for roll control | kdR | uint16 | [100/sec] |
| P-value for pitch control | kpP | uint16 | [100/sec2] |
| I-value for pitch control | kiP | uint16 | [100/sec3] |
| D-value for pitch control | kdP | uint16 | [100/sec] |
| P-value for yaw control | kpY | uint16 | [100/sec2] |
| I-value for yaw control | kiY | uint16 | [100/sec3] |
| D-value for yaw control | kdY | uint16 | [100/sec] |
| Low pass filter co-efficient (roll angle) | lpR | int16 | [] |
| High pass filter co-efficient (roll angle) | hpR | int16 | [] |
| Low pass filter co-efficient (pitch angle) | lpP | int16 | [] |
| High pass filter co-efficient (pitch angle) | hpP | int16 | [] |
| Inertia around the x-axis | inertiaX | int16 | [1000\*N\*m\*sec2] |
| Inertia around the y-axis | inertiaY | int16 | [1000\*N\*m\*sec2] |
| Inertia around the z-axis | inertiaZ | int16 | [1000\*N\*m\*sec2] |
| Length of the booms | lenBoom | int16 | [mm] |
| Maximum desirable force | forceMax | uint16 | [1000\*N] |
| Maximum desirable yaw velocity | angVelYMax | uint16 | [(1000\*rad)/sec] |
| Maximum desirable pitch angle | angPMax | uint16 | [1000\*rad] |
| Maximum desirable roll angle | angRMax | uint16 | [1000\*rad] |

Table 2: Calibration parameter

# System states

The table below shows the states of the quadrocopter, transmitted via XBee to the base station for observation and visualization purposes.

|  |  |  |  |
| --- | --- | --- | --- |
| **State** | **Abbreviation** | **Data Type** | **Range / Unit** |
| Acceleration in X direction (sensor) | accXRaw | int16 | -2048..2047 |
| Acceleration in X direction (physical) | accX | int16 | [(10\*m)/sec2] |
| Acceleration in Y direction (sensor) | accYRaw | int16 | -2048..2047 |
| Acceleration in Y direction (physical) | accY | int16 | [(10\*m)/sec2] |
| Acceleration in Z direction (sensor) | accZRaw | int16 | -2048..2047 |
| Acceleration in Z direction (physical) | accZ | int16 | [(10\*m)/sec2] |
| Roll velocity (sensor) | angVelRRaw | int16 | 0..1024 |
| Roll velocity (physical) | angVelR | int16 | [(100\*rad)/sec] |
| Roll angle (physical) | angR | int16 | [10000\*rad] |
| Pitch velocity (sensor) | angVelPRaw | int16 | 0..1024 |
| Pitch velocity (physical) | angVelP | int16 | [(100\*rad)/sec] |
| Pitch angle (physical) | angP | int16 | [10000\*rad] |
| Yaw velocity (sensor) | angVelYRaw | int16 | 0..1024 |
| Yaw velocity (physical) | angVelY | int16 | [(100\*rad)/sec] |
| Temperature (sensor) | tempRaw | int16 | 0..1024 |
| Temperature (physical) | temp | uint8 | [°C] |
| Air pressure (sensor) | airPressureRaw | int16 | 0..1024 |
| Battery voltage (sensor) | batteryRaw | int16 | 0..1024 |
| Battery voltage (physical) | battery | uint8 | [10\*V] |
| Speed front rotor (physical) | rpmFront | uint16 | [1/min] |
| Speed left rotor (physical) | rpmLeft | uint16 | [1/min] |
| Speed rear rotor (physical) | rpmRear | uint16 | [1/min] |
| Speed right rotor (physical) | rpmRight | uint16 | [1/min] |
| Total motor force/thrust (physical) | forceTotal | uint16 | [1000\*N] |
| Remote control connected flag | remoteConnected | bool | 0..1 |
| Motors on flag | remoteMotorsOn | bool | 0..1 |
| Desired force/thrust (remote) | remoteForceRaw | uint8 | 0..255 |
| Desired force/thrust (physical) | remoteForce | uint16 | [1000\*N] |
| Desired yaw velocity (remote) | remoteYawRaw | uint8 | 0..255 |
| Desired yaw velocity (physical) | remoteYaw | int16 | [(1000\*rad)/sec] |
| Desired pitch angle (remote) | remotePitchRaw | uint8 | 0..255 |
| Desired pitch angle (physical) | remotePitch | int16 | [1000\*rad] |
| Desired roll angle (remote) | remoteRollRaw | uint8 | 0..255 |
| Desired roll angle (physical) | remoteRoll | int16 | [1000\*rad] |

Table 3: System states

# Communication / Messages

**Communication principles:**

* The communication is byte oriented.
* The base station initiates all communication.
* For transmission error detection and synchronization purposes, a CRC16 checksum is added to each message (generator polynomial: 0xA001).
* The MSB (Most Significant Bit) of a byte/value is always sent first. This holds also for 16-bit values (Bits: [16…8] [7…0]).
* A message consists out of an identifier, data length field, data field and checksum. The maximum message length is 256 byte (252 byte payload). For details, please see ‘Basic message layout’ paragraph.
* There must be a gap of at least 20ms between two consecutive messages, sent from the base station. Otherwise messages are lost in the flight control, because of a buffer overflow/overwrite. The gap between two bytes of a message must not be larger than 2ms.

**Basic message layout:**

First, an one byte ID is transmitted, specifying the message type. The second byte represents the number of payload bytes (Data Length Code - DLC). The payload itself can vary between zero and 252 bytes. A 16-bit checksum (Cyclic Redundancy Check - CRC) finishes the message. The basic layout is illustrated in figure 3.1. Each block represents one byte.

The content of the ID and data field is described for each message separately. The CRC is calculated over the ID, DLC and the data field.

…

Data1

DLC

ID

CRC

CRC

DataX

Figure .1: General message layout

**Physical data transmission**

For communication with the XBee modules over UART (RS232) the following configuration is used:

* 9600 baud/sec
* 1 start bit, 8 data bits, 1 stop bit
* No parity

All messages, which are sent between quadrocopter and base station, are defined on the next pages. Other messages will be ignored.

*CAUTION*: In general it must be ensured, that the communication between quadrocopter and base station has no bad influence on the flight control task. E.g. the transmission of a message from the quadrocopter to the base station blocks the flight control task longer than its calculation period.

*CAUTION*: Because the wireless link between the XBee modules is a half-duplex communication, it must be ensured, that both sides are able to transmit data. Neither the quadrocopter nor the base station is allowed to occupy the wireless channel completely by sending data all the time.

## Parameter Request (base station 🡪 quadrocopter)

This message is sent from the base station in order to request the actual parameter set of the quadrocopter. This includes all controller parameter as well as the parameters for the sensor filter. When the quadrocopter receives this message, it has to respond with a ‘Parameter Response’ message (see chapter 3.2).

ID: 0x10

DLC: 0x00 (0)

Data: None

## Parameter Response (quadrocopter 🡪 base station)

This message is sent from the quadrocopter, when a ‘Parameter Request’ or a ‘Parameter Update’ message was received. The data field contains the actual controller and sensor filter parameters in the same order as table … shows.

ID: 0x11

DLC: 2A (42)

Data: All controller and sensor filter parameter; for ordering see ‘Table 2’

## Parameter Update (base station 🡪 quadrocopter)

This message is sent from the base station in order to update the controller and sensor filter parameters of the quadrocopter flight control. The message will only have an effect, if the quadrocopter is at the ground (rpm of all motors equal to zero). The data field contains the new controller and sensor filter parameters in the same order as in ‘Table 2’. When the quadrocopter receives this message, it has to respond with a ‘Parameter Response’ message (see chapter 3.2). When it is up in the air, the actual parameters are sent back.

ID: 0x12

DLC: 2A (42)

Data: All controller and sensor filter parameter; for ordering see ‘Table 2’

## Status Request (base station 🡪 quadrocopter)

This message is sent from the base station in order to request the actual system status. Dependent on the data byte, the quadrocopter either responds only once or sends periodically a ‘Status Response’ message (see chapter 3.5). A data byte value of 0xFF requests a single response message. For all other values, the data byte specifies the response period. Response period = data byte value \* 10ms. Sending 0x00 in the data field stops a periodical response of the quadrocopter.

ID: 0x20

DLC: 0x01 (1)

Data: One byte: 0x00 – stop periodical response

0x0A – ‘Status Response’ message every 100ms

0x14 – ‘Status Response’ message every 200ms

0x32 – ‘Status Response’ message every 500ms

0x64 – ‘Status Response’ message every 1000ms

0xFF – single ‘Status Response’ message

*There are only some examples for the data byte value listed above. All other values in between are also possible.*

## Status Response (quadrocopter 🡪 base station)

This message is sent from the quadrocopter, when a ‘Status Request’ was received, either once or periodically (see ‘Status Request’, chapter 3.4). The data field contains the actual status/sensor values in the same order as table … shows.

ID: 0x21

DLC: 0x3C (60)

Data: All actual status/sensor values; for ordering see ‘Table 3’

# Embedded Software

## General Structure

The embedded software, running on the HCS12X microcontroller, follows the time-triggered approach and is divided in three layers: The Application Layer, an intermediate layer, containing the real-time image and the Hardware Abstraction Layer (HAL). The structure is shown in figure X.1.

Hardware Abstraction Layer (HAL)

QH\_eeprom

QH\_xbee

QH\_beeper

QH\_accelerometer

QH\_atd

QH\_timer

QH\_brushless

QH\_remote

QH\_pll

QH\_led

Real-Time Image Layer

Copter

Application Layer

basestation

typedef

flightcontrol

sensorfilter

main

Figure .

The Hardware Abstraction Layer (HAL) encapsulates all functions, which are directly related to the microcontroller, sensors and actuators. This implies, that in the layers above, there is no CPU register used directly. Except of the timer module (OH\_timer) all functions of the HAL are only by the real-time image layer. The application layer uses the timer functionality directly to organize scheduling of the tasks.

## Hardware Abstraction Layer

## Real-Time Image Layer

## Application Layer

### Tasks

As mentioned above the system is time-triggered and therefore it uses time-triggered tasks only.

The most important task of the software is of course the flight control. The control algorithm is calculated every ten milliseconds. Before the control algorithm is executed, the real-time image is actualized and after calculating the controller output signal, the actuator setpoint is adapted.

For telemetry purposes, the quadrocopter is equipped with a XBee radio module. The Software checks every 100 milliseconds if there is a new service request by the base station or if data has to be sent back.

In a period of one second, a diagnosis task is scheduled checking the basic parameters and functionalities of the quadrocopter.