# xyControl

0.1

Generated by Doxygen 1.8.3.1

Tue Apr 2 2013 20:43:30

# Contents

# Chapter 1

# Main Page

xyControl is a Quadrocopter Flight Controller based on Atmels Atmega2560 microcontroller. It features 64KB SRAM on-board, using the external memory interface of this processor. Also included is a switched power supply as well as a USB connection to communicate with and program the target. All I/O pins, including 3 additional UARTs, SPI, I2C (TWI) and 16 ADC Channels, are accessible via standard 2.54mm connectors. The Board can be powered from an external stable 5V supply, USB or 7V or more, via the on-board switched power supply. All voltage sources can be selected via jumpers.

### Known Problems

In the current PCB layout, the SD-Card holder is rotated 180 degrees. This prevents an SD-Card from being inserted!

### Flight Control Software Flow

Three tasks are controlling the Quadrocopter Orientation in Space.

- The Orientation Task reads the Gyroscope and Accelerometer and calculates the current Roll and Pitch angles. They are stored in the global struct "orientation".
- The PID Task is then feeding these angles into two PID controllers. Their output is then used by...
- The Set Task, which calculates the motor speeds and gives them to...
- The motor task, which sends the new values via TWI to the motor controllers.

## Supported Hardware

- Gyroscope L3GD20, code based on the `Adafruit Example`.
- Accelerometer and Magnetometer LSM303DLHC, code based on the `Pololu Example`.
- I got both of these Sensors on the `MinIMU-9 v2`
- Brushless Motor Driver `BL-Ctrl V1.2` with eg. the `Robbe Roxxy Outrunner 2824-34` Brushless Motor.
- BTM-222 Bluetooth UART Bridge (`PCB`)

### External Memory (xmem.h)

The external memory consists of a 512Kx8 SRAM, bank-switched onto the 16bit avr address space. This gives us 8 memory banks, consisting of 56KB. All memory from 0x0000 to 0x21FF is the AVRs internal memory. The memory banks are switched into 0x2200 to 0xFFFF. This gives us 8 banks with 56KB each, resulting in 448KB external RAM.

The data and bss memory sections, as well as the Stack are located in the internal RAM. The external RAM is used only for dynamically allocated memory.

### Orientation Calculation (orientation.h)

Calculates the current angles of the platform, using Gyroscope and Accelerometer Data with a Kalman Filter. It is using this slightly modified Kalman Filter Implementation by Linus Helgesson.

## PC and Android Tools

You can find some PC Software in the 'tools' directory. Each one should be accompanied by it's own Readme file.

### UART-Flight Status Packet Format

```
printf("t%.2f %.2f %.2f\n", kp, ki, kd);
printf("u%.2f %.2f\n", pid_output[1], pid_output[0]); // Pitch, Roll
printf("v%i %i %i %i\n", motorSpeed[0], ..., motorSpeed[3]);
printf("w%.2f\n", orientation.pitch);
printf("x%.2f\n", orientation.roll);
printf("y%.2f\n", orientation.yaw);
printf("z%.2f\n", getVoltage());
```

## Software used

- Peter Fleurys TWI Library

## License

Peter Fleurys TWI Library (twi.c & twi.h) is released under the GNU GPL license.

Everything else is released under a BSD-Style license. See the accompanying COPYING file.

# Chapter 2

# Module Index

## 2.1 Modules

Here is a list of all modules:

# Chapter 3

# Data Structure Index

## 3.1 Data Structures

Here are the data structures with brief descriptions:

# Chapter 4

# File Index

## 4.1 File List

Here is a list of all documented files with brief descriptions:

# Chapter 5

# Module Documentation

## 5.1 Software

Software Libraries.

### Modules

- System

  *System Libraries.*
- Flight

  *Flight Control Libraries.*

### 5.1.1 Detailed Description

Software Libraries.

## 5.2 System

System Libraries.

### Modules

- Debug Output

  *Allows debug ouput and assert usage.*
- Error Reporting

  *Error reporting with human readable strings.*
- Task Handler

  *System for registering different tasks that will be called regularly, one after another.*
- Time Keeping

  *Measuring Time with Millisecond Resolution.*
- UART Menu

  *Enables user interaction with an UART Menu.*
- External Memory Interface

  *Allows access to external RAM with bank-switching.*
- xyControl Hardware

  *Controls xyControl On-Board Hardware like LEDs.*

### 5.2.1 Detailed Description

System Libraries.

## 5.3 Flight

Flight Control Libraries.

### Modules

- Kalman-Filter

  *Kalman-Filter from* `Linus Helgesson`
- Orientation Calculation

  *Calculate Orientation using the Kalman-Filter, Accelerometer and Gyroscope.*
- PID-Controller

  *Simple implementation for multiple floating-point PID Controllers.*
- Motor Speed Mixer

  *Takes the Base Speed and PID-Output and sets Motor Speed accordingly.*

### 5.3.1 Detailed Description

Flight Control Libraries.

## 5.4 Hardware

Hardware Libraries.

### Modules

- Accelerometer Driver

  *Configuring and reading an LSM303DLHC Accelerometer.*
- ADC Driver

  *Analog-to-Digital Converter Library.*
- Gyroscope Driver

  *Configuring and reading an L3GD20.*
- Magnetometer Driver

  *Configuring and reading an LSM303DLHC Magnetometer.*
- Motor Controller Driver

  *Controlling four* `BL-Ctrl V1.2` *Brushless controllers.*
- UART Driver

  *Serial Library for AVRs built-in UART Hardware.*
- SPI Driver

  *SPI Library for AVRs built-in SPI Hardware.*
- I2C Driver

  *Using the AVR TWI/I2C Hardware.*

### 5.4.1 Detailed Description

Hardware Libraries.

## 5.5 Accelerometer Driver

Configuring and reading an LSM303DLHC Accelerometer.

### Files

- file acc.h
  *LSM303DLHC Accelerometer API Header.*
- file acc.c
  *LSM303DLHC Accelerometer API Implementation.*

### Macros

- #define ACCREG_CTRL1 0x20
  *Accelerometer Control Register 1.*
- #define ACCREG_CTRL4 0x23
  *Accelerometer Control Register 4.*
- #define ACCREG_XL 0x28
  *First Accelerometer Output Register.*

### Enumerations

- enum AccRange { r2G, r4G, r8G, r16G }
  *Accelerometer Range options.*

### Functions

- Error accInit (AccRange r)
  *Initialize the Accelerometer.*
- Error accRead (Vector3f *v)
  *Read from the Accelerometer.*
- Error accWriteRegister (uint8_t reg, uint8_t val)
  *Write an Accelerometer Register.*

### Variables

- AccRange accRange
  *Stored range to scale returned values.*

### 5.5.1 Detailed Description

Configuring and reading an LSM303DLHC Accelerometer.

### 5.5.2 Macro Definition Documentation

#### 5.5.2.1 #define ACCREG_CTRL1 0x20

Accelerometer Control Register 1.

Definition at line 49 of file acc.c.

Referenced by accInit().

#### 5.5.2.2 #define ACCREG_CTRL4 0x23

Accelerometer Control Register 4.

Definition at line 50 of file acc.c.

Referenced by accInit().

#### 5.5.2.3 #define ACCREG_XL 0x28

First Accelerometer Output Register.

Definition at line 51 of file acc.c.

Referenced by accRead().

### 5.5.3 Enumeration Type Documentation

#### 5.5.3.1 enum AccRange

Accelerometer Range options.

**Enumerator**

***r2G*** +- 2G

***r4G*** +- 4G

***r8G*** +- 8G

***r16G*** +- 16G

Definition at line 47 of file acc.h.

```
47              {
48     r2G,
49     r4G,
50     r8G,
51     r16G,
52 } AccRange;
```

### 5.5.4 Function Documentation

#### 5.5.4.1 Error accInit ( AccRange *r* )

Initialize the Accelerometer.

Call before accRead(). I2C should already be initialized!

**Parameters**

| | |
|---|---|
| *r* | AccRange to use. |

**Returns**

TWI_NO_ANSWER, TWI_WRITE_ERROR, ARGUMENT_ERROR or SUCCESS.

Definition at line 76 of file acc.c.

References accRange, ACCREG_CTRL1, ACCREG_CTRL4, accWriteRegister(), ARGUMENT_ERROR, r16G, r2G, r4G, r8G, and SUCCESS.

Referenced by orientationInit().

```
76                              {
77      uint8_t v;
78      switch (r) {
79          case r2G:
80              v = 0x00;
81              break;
82          case r4G:
83              v = 0x10;
84              break;
85          case r8G:
86              v = 0x20;
87              break;
88          case r16G:
89              v = 0x30;
90              break;
91          default:
92              return ARGUMENT_ERROR;
93      }
94      accRange = r;
95      Error e = accWriteRegister(ACCREG_CTRL1, 0x57); // Enable all axes,
        100Hz
96      if (e != SUCCESS) {
97          return e;
98      }
99      e = accWriteRegister(ACCREG_CTRL4, v);
100      return e;
101  }
```

#### 5.5.4.2 Error accRead ( Vector3f * v )

Read from the Accelerometer.

Accelerometer should already be initialized!

**Parameters**

| | |
|---|---|
| *v* | Vector3f for the read values |

**Returns**

TWI_NO_ANSWER, TWI_WRITE_ERROR, ARGUMENT_ERROR or SUCCESS.

Definition at line 103 of file acc.c.

References ACC_ADDRESS, ACCFILTERFACTOR, accRange, ACCREG_XL, ARGUMENT_ERROR, r16G, r2G, r4G, r8G, SUCCESS, TWI_NO_ANSWER, TWI_READ, TWI_WRITE, TWI_WRITE_ERROR, twiReadAck(), twiReadNak(), twiRepStart(), twiStart(), twiWrite(), Vector3f::x, Vector3f::y, and Vector3f::z.

Referenced by orientationTask().

```
103                               {
104      static double accSumX = 0; /* Buffer for X Low-Pass. */
105      static double accSumY = 0; /* Buffer for Y Low-Pass. */
106      static double accSumZ = 0; /* Buffer for Z Low-Pass. */
107      static double accFilterX = 0; /* Buffer for X Low-Pass. */
108      static double accFilterY = 0; /* Buffer for Y Low-Pass. */
109      static double accFilterZ = 0; /* Buffer for Z Low-Pass. */
110
111      if (v == NULL) {
112          return ARGUMENT_ERROR;
113      }
114      if (twiStart(ACC_ADDRESS | TWI_WRITE)) {
115          return TWI_NO_ANSWER;
116      }
117      if (twiWrite(ACCREG_XL | (1 << 7))) { // Auto Increment
118          return TWI_WRITE_ERROR;
119      }
120      if (twiRepStart(ACC_ADDRESS | TWI_READ)) {
121          return TWI_NO_ANSWER;
122      }
123
124      uint8_t xl = twiReadAck();
125      uint8_t xh = twiReadAck();
126      uint8_t yl = twiReadAck();
127      uint8_t yh = twiReadAck();
```

```
128     uint8_t zl = twiReadAck();
129     uint8_t zh = twiReadNak();
130 
131     int16_t x = *(int8_t *)(&xh);
132     x *= (1 << 8);
133     x |= xl;
134 
135     int16_t y = *(int8_t *)(&yh);
136     y *= (1 << 8);
137     y |= yl;
138 
139     int16_t z = *(int8_t *)(&zh);
140     z *= (1 << 8);
141     z |= zl;
142 
143     switch (accRange) {
144         case r2G:
145             v->x = (((double)x) * 2 / 0x8000);
146             v->y = (((double)y) * 2 / 0x8000);
147             v->z = (((double)z) * 2 / 0x8000);
148             break;
149         case r4G:
150             v->x = (((double)x) * 4 / 0x8000);
151             v->y = (((double)y) * 4 / 0x8000);
152             v->z = (((double)z) * 4 / 0x8000);
153             break;
154         case r8G:
155             v->x = (((double)x) * 8 / 0x8000);
156             v->y = (((double)y) * 8 / 0x8000);
157             v->z = (((double)z) * 8 / 0x8000);
158             break;
159         case r16G:
160             v->x = (((double)x) * 16 / 0x8000);
161             v->y = (((double)y) * 16 / 0x8000);
162             v->z = (((double)z) * 16 / 0x8000);
163             break;
164         default:
165             return ARGUMENT_ERROR;
166     }
167 
168     accSumX = accSumX - accFilterX + v->x;
169     accFilterX = accSumX / ACCFILTERFACTOR;
170     v->x = accFilterX;
171 
172     accSumY = accSumY - accFilterY + v->y;
173     accFilterY = accSumY / ACCFILTERFACTOR;
174     v->y = accFilterY;
175 
176     accSumZ = accSumZ - accFilterZ + v->z;
177     accFilterZ = accSumZ / ACCFILTERFACTOR;
178     v->z = accFilterZ;
179 
180     return SUCCESS;
181 }
```

#### 5.5.4.3 Error accWriteRegister ( uint8_t *reg*, uint8_t *val* )

Write an Accelerometer Register.

I2C should aready be initialized!

**Parameters**

| | |
|---|---|
| *reg* | Register Address |
| *val* | New Value |

**Returns**

TWI_NO_ANSWER, TWI_WRITE_ERROR or SUCCESS.

Definition at line 62 of file acc.c.

References TWI_NO_ANSWER.

Referenced by accInit().

```
62                                                  {
```

```
63      if (twiStart(ACC_ADDRESS | TWI_WRITE)) {
64          return TWI_NO_ANSWER;
65      }
66      if (twiWrite(reg)) {
67          return TWI_WRITE_ERROR;
68      }
69      if (twiWrite(val)) {
70          return TWI_WRITE_ERROR;
71      }
72      twiStop();
73      return SUCCESS;
74  }
```

### 5.5.5 Variable Documentation

#### 5.5.5.1 AccRange accRange

Stored range to scale returned values.

Definition at line 53 of file acc.c.

Referenced by accInit(), and accRead().

## 5.6 ADC Driver

Analog-to-Digital Converter Library.

### Files

- file adc.h
  *Analog-to-Digital Converter API Header.*
- file adc.c
  *Analog-to-Digital Converter API Implementation.*

### Enumerations

- enum ADCRef { AREF, AVCC, AINT1, AINT2 }
  *ADC Reference Voltage options.*

### Functions

- void adcInit (ADCRef ref)
  *Initialize the ADC Hardware.*
- void adcStart (uint8_t channel)
  *Start a conversion on a given channel.*
- uint8_t adcReady (void)
  *Check if a result is ready.*
- uint16_t adcGet (uint8_t next)
  *Get the conversion results.*
- void adcClose (void)
  *Disable the ADC to save energy.*

### 5.6.1 Detailed Description

Analog-to-Digital Converter Library. With 10bit Output and selectable Reference Voltage.

### 5.6.2 Enumeration Type Documentation

#### 5.6.2.1 enum ADCRef

ADC Reference Voltage options.

**Enumerator**

***AREF*** External Reference Voltage.

***AVCC*** Supply Voltage.

***AINT1*** Internal Reference 1 (1.1V)

***AINT2*** Internal Reference 2 (2.56V)

Definition at line 45 of file adc.h.

```
45              {
46     AREF,
47     AVCC,
48     AINT1,
49     AINT2
50 } ADCRef;
```

### 5.6.3 Function Documentation

#### 5.6.3.1 void adcClose ( void )

Disable the ADC to save energy.

Definition at line 107 of file adc.c.

```
107                 {
108     // deactivate adc
109     ADCSRA &= ~(1 << ADSC);
110     PRR0 |= (1 << PRADC);
111     ADCSRA &= ~(1 << ADEN);
112 }
```

#### 5.6.3.2 uint16_t adcGet ( uint8_t *next* )

Get the conversion results.

**Parameters**

| | |
|---|---|
| *next* | Start next conversion if != 0 |

**Returns**

10bit ADC value

Definition at line 96 of file adc.c.

References adcReady().

Referenced by getVoltage().

```
96                              {
97     // Return measurements result
98     // Start next conversion
99     uint16_t temp = 0;
100     while (!adcReady());
101     temp = ADC;
102     if (next)
103         ADCSRA |= (1 << ADSC); // Start next conversion
104     return temp;
105 }
```

#### 5.6.3.3 void adcInit ( ADCRef *ref* )

Initialize the ADC Hardware.

**Parameters**

| | |
|---|---|
| *ref* | Reference Voltage. |

Definition at line 44 of file adc.c.

References AINT1, AINT2, AREF, and AVCC.

Referenced by xyInit().

```
44                          {
45     // Enable ADC Module, start one conversion, wait for finish
46     PRR0 &= ~(1 << PRADC); // Disable ADC Power Reduction (Enable it...)
47     switch(ref) {
48         case AVCC:
49             ADMUX = (1 << REFS0);
50             break;
```

```
51
52          case AINT1:
53              ADMUX = (1 << REFS1);
54              break;
55
56          case AINT2:
57              ADMUX = (1 << REFS1) | (1 << REFS0);
58              break;
59
60          case AREF:
61              ADMUX &= ~((1 << REFS0) | (1 << REFS1));
62              break;
63      }
64
65      ADCSRA = (1 << ADPS2) | (1 << ADPS1) | (1 << ADPS0); // Prescaler 128
66      ADCSRB = 0;
67      ADCSRA |= (1 << ADEN) | (1 << ADSC); // Start ADC, single conversion
68  }
```

#### 5.6.3.4 uint8_t adcReady ( void )

Check if a result is ready.

**Returns**

1 if conversion is done.

Definition at line 86 of file adc.c.

Referenced by adcGet(), and getVoltage().

```
86                      {
87      // Is the measurement finished
88      if (ADCSRA & (1 << ADSC)) {
89          // ADSC bit is set
90          return 0;
91      } else {
92          return 1;
93      }
94  }
```

#### 5.6.3.5 void adcStart ( uint8_t *channel* )

Start a conversion on a given channel.

**Parameters**

| | |
|---|---|
| *channel* | Channel (0 - 15) |

Definition at line 70 of file adc.c.

Referenced by getVoltage().

```
70                                  {
71      // Start a measurement on channel
72      if (channel > 15) {
73          channel = 0;
74      }
75      if (channel > 7) {
76          channel -= 8;
77          ADCSRB |= (1 << MUX5);
78      } else {
79          ADCSRB &= ~(1 << MUX5);
80      }
81      ADMUX &= ~0x1F; // Delete MUX0:4
82      ADMUX |= channel;
83      ADCSRA |= (1 << ADSC);
84  }
```

## 5.7 Configuration

Various default settings.

### Files

- file config.h

  *Various default settings.*

### Macros

- #define SOFTWARELOWPASS 50

  *Software Low-Pass on Gyro and ACC.*
- #define ACCFILTERFACTOR SOFTWARELOWPASS

  *Accelerometer Low Pass Factor.*
- #define GYROFILTERFACTOR SOFTWARELOWPASS

  *Gyroscope Low Pass Factor.*
- #define PID_OUTMAX 256

  *Maximum PID Output.*
- #define PID_OUTMIN -256

  *Minimum PID Output.*
- #define PID_INTMAX PID_OUTMAX

  *Maximum PID Integral Sum.*
- #define PID_INTMIN PID_OUTMIN

  *Minimal PID Integral Sum.*
- #define PID_FACTOR 4 / 5

  *Influence of PID in relation to Base Speed.*
- #define DT 0.01f

  *Time Constant.*
- #define Q1 5.0f

  *Q Matrix Diagonal Element 1.*
- #define Q2 100.0f

  *Q Matrix Diagonal Element 2.*
- #define Q3 0.01f

  *Q Matrix Diagonal Element 3.*
- #define R1 1000.0f

  *R Matrix Diagonal Element 1.*
- #define R2 1000.0f

  *R Matrix Diagonal Element 2.*
- #define SET_ROLLPLUS 1

  *Second Motor at the Right.*
- #define SET_ROLLMINUS 3

  *Fourth Motor at the Left.*
- #define SET_PITCHPLUS 0

  *First Motor at the Top.*
- #define SET_PITCHMINUS 2

  *Third Motor at the Bottom.*
- #define PID_P 5.0

  *Default PID P Constant.*
- #define PID_I 0.03

*Default PID I Constant.*

- #define PID_D -13.0

  *Default PID D Constant.*

- #define MOTORCOUNT 4

  *Amount of motors.*

- #define BATT_MAX 15

  *Battery Voltage Reference (ADC 5V)*

- #define BATT_CHANNEL 0

  *ADC Channel for Battery.*

- #define ACC_ADDRESS 0x32

  *Accelerometer Address (0011001r)*

- #define GYRO_ADDRESS 0xD6

  *Gyroscope Address (110101xr, x = 1)*

- #define MAG_ADDRESS 0x3C

  *Magnetometer Address.*

- #define MOTOR_BASEADDRESS 0x52

  *Address of first motor controller.*

- #define LED0PORT PORTL

  *First LED Port.*

- #define LED0DDR DDRL

  *First LED Data Direction Register.*

- #define LED0PIN PL6

  *First LED Pin.*

- #define LED1PORT PORTL

  *Second LED Port.*

- #define LED1DDR DDRL

  *Second LED Data Direction Register.*

- #define LED1PIN PL7

  *Second LED Pin.*

- #define LED2PORT PORTG

  *Third LED Port.*

- #define LED2DDR DDRG

  *Third LED Data Direction Register.*

- #define LED2PIN PG5

  *Third LED Pin.*

- #define LED3PORT PORTE

  *Fourth LED Port.*

- #define LED3DDR DDRE

  *Fourth LED Data Direction Register.*

- #define LED3PIN PE2

  *Fourth LED Pin.*

- #define BANK0PORT PORTG

  *First Bank Selection Port.*

- #define BANK0DDR DDRG

  *First Bank Selection Data Direction Register.*

- #define BANK0PIN PG3

  *First Bank Selection Pin.*

- #define BANK1PORT PORTG

  *Second Bank Selection Port.*

- #define BANK1DDR DDRG

  *Second Bank Selection Data Direction Register.*

- #define BANK1PIN PG4

  *Second Bank Selection Pin.*
- #define BANK2PORT PORTL

  *Third Bank Selection Port.*
- #define BANK2DDR DDRL

  *Third Bank Selection Data Direction Register.*
- #define BANK2PIN PL5

  *Third Bank Selection Pin.*
- #define SPISS PB0

  *SPI Slave Select Pin.*
- #define RX_BUFFER_SIZE 128

  *UART Receive Buffer Size.*
- #define TX_BUFFER_SIZE 128

  *UART Transmit Buffer Size.*
- #define USB 0

  *FT232RL USB UART module.*
- #define BLUETOOTH 1

  *Bluetooth UART module.*
- #define UART BLUETOOTH

  *UART module to use (0 to 3)*

### 5.7.1 Detailed Description

Various default settings.

### 5.7.2 Macro Definition Documentation

#### 5.7.2.1 #define ACC_ADDRESS 0x32

Accelerometer Address (0011001r)

Definition at line 108 of file config.h.

Referenced by accRead().

#### 5.7.2.2 #define ACCFILTERFACTOR SOFTWARELOWPASS

Accelerometer Low Pass Factor.

Definition at line 47 of file config.h.

Referenced by accRead().

#### 5.7.2.3 #define BANK0DDR DDRG

First Bank Selection Data Direction Register.

Definition at line 135 of file config.h.

Referenced by xmemInit().

#### 5.7.2.4 #define BANK0PIN PG3

First Bank Selection Pin.

Definition at line 136 of file config.h.

Referenced by xmemInit(), and xmemSetBank().

#### 5.7.2.5 #define BANK0PORT PORTG

First Bank Selection Port.

Definition at line 134 of file config.h.

Referenced by xmemInit(), and xmemSetBank().

#### 5.7.2.6 #define BANK1DDR DDRG

Second Bank Selection Data Direction Register.

Definition at line 138 of file config.h.

Referenced by xmemInit().

#### 5.7.2.7 #define BANK1PIN PG4

Second Bank Selection Pin.

Definition at line 139 of file config.h.

Referenced by xmemInit(), and xmemSetBank().

#### 5.7.2.8 #define BANK1PORT PORTG

Second Bank Selection Port.

Definition at line 137 of file config.h.

Referenced by xmemInit(), and xmemSetBank().

#### 5.7.2.9 #define BANK2DDR DDRL

Third Bank Selection Data Direction Register.

Definition at line 141 of file config.h.

Referenced by xmemInit().

#### 5.7.2.10 #define BANK2PIN PL5

Third Bank Selection Pin.

Definition at line 142 of file config.h.

Referenced by xmemInit(), and xmemSetBank().

#### 5.7.2.11 #define BANK2PORT PORTL

Third Bank Selection Port.

Definition at line 140 of file config.h.

Referenced by xmemInit(), and xmemSetBank().

#### 5.7.2.12 #define BATT_CHANNEL 0

ADC Channel for Battery.

Definition at line 102 of file config.h.

Referenced by getVoltage().

#### 5.7.2.13 #define BATT_MAX 15

Battery Voltage Reference (ADC 5V)

Definition at line 101 of file config.h.

Referenced by getVoltage().

#### 5.7.2.14 #define BLUETOOTH 1

Bluetooth UART module.

Definition at line 167 of file config.h.

#### 5.7.2.15 #define DT 0.01f

Time Constant.

Definition at line 63 of file config.h.

Referenced by kalmanInnovate().

#### 5.7.2.16 #define GYRO_ADDRESS 0xD6

Gyroscope Address (110101xr, x = 1)

Definition at line 109 of file config.h.

Referenced by gyroRead().

#### 5.7.2.17 #define GYROFILTERFACTOR SOFTWARELOWPASS

Gyroscope Low Pass Factor.

Definition at line 48 of file config.h.

Referenced by gyroRead().

#### 5.7.2.18 #define LED0DDR DDRL

First LED Data Direction Register.

Definition at line 118 of file config.h.

Referenced by xyInit().

#### 5.7.2.19 #define LED0PIN PL6

First LED Pin.

Definition at line 119 of file config.h.

Referenced by xyInit().

#### 5.7.2.20 #define LED0PORT PORTL

First LED Port.

Definition at line 117 of file config.h.

#### 5.7.2.21 #define LED1DDR DDRL

Second LED Data Direction Register.

Definition at line 121 of file config.h.

Referenced by xyInit().

#### 5.7.2.22 #define LED1PIN PL7

Second LED Pin.

Definition at line 122 of file config.h.

Referenced by xyInit().

#### 5.7.2.23 #define LED1PORT PORTL

Second LED Port.

Definition at line 120 of file config.h.

#### 5.7.2.24 #define LED2DDR DDRG

Third LED Data Direction Register.

Definition at line 124 of file config.h.

Referenced by xyInit().

#### 5.7.2.25 #define LED2PIN PG5

Third LED Pin.

Definition at line 125 of file config.h.

Referenced by xyInit().

#### 5.7.2.26 #define LED2PORT PORTG

Third LED Port.

Definition at line 123 of file config.h.

#### 5.7.2.27 #define LED3DDR DDRE

Fourth LED Data Direction Register.

Definition at line 127 of file config.h.

Referenced by xyInit().

#### 5.7.2.28 #define LED3PIN PE2

Fourth LED Pin.

Definition at line 128 of file config.h.

Referenced by xyInit().

#### 5.7.2.29 #define LED3PORT PORTE

Fourth LED Port.

Definition at line 126 of file config.h.

#### 5.7.2.30 #define MAG_ADDRESS 0x3C

Magnetometer Address.

Definition at line 110 of file config.h.

Referenced by magRead().

#### 5.7.2.31 #define MOTOR_BASEADDRESS 0x52

Address of first motor controller.

Definition at line 111 of file config.h.

Referenced by motorTask().

#### 5.7.2.32 #define MOTORCOUNT 4

Amount of motors.

Definition at line 95 of file config.h.

Referenced by motorInit(), motorSet(), and motorTask().

#### 5.7.2.33 #define PID_D -13.0

Default PID D Constant.

Definition at line 89 of file config.h.

Referenced by pidInit().

#### 5.7.2.34 #define PID_FACTOR 4 / 5

Influence of PID in relation to Base Speed.

Definition at line 56 of file config.h.

#### 5.7.2.35 #define PID_I 0.03

Default PID I Constant.

Definition at line 88 of file config.h.

Referenced by pidInit().

#### 5.7.2.36 #define PID_INTMAX PID_OUTMAX

Maximum PID Integral Sum.

Definition at line 52 of file config.h.

Referenced by pidInit().

#### 5.7.2.37 #define PID_INTMIN PID_OUTMIN

Minimal PID Integral Sum.

Definition at line 53 of file config.h.

Referenced by pidInit().

#### 5.7.2.38 #define PID_OUTMAX 256

Maximum PID Output.

Definition at line 50 of file config.h.

Referenced by pidInit().

#### 5.7.2.39 #define PID_OUTMIN -256

Minimum PID Output.

Definition at line 51 of file config.h.

Referenced by pidInit().

#### 5.7.2.40 #define PID_P 5.0

Default PID P Constant.

Definition at line 87 of file config.h.

Referenced by pidInit().

#### 5.7.2.41 #define Q1 5.0f

Q Matrix Diagonal Element 1.

Definition at line 66 of file config.h.

Referenced by kalmanInnovate().

#### 5.7.2.42 #define Q2 100.0f

Q Matrix Diagonal Element 2.

Definition at line 67 of file config.h.

Referenced by kalmanInnovate().

#### 5.7.2.43 #define Q3 0.01f

Q Matrix Diagonal Element 3.

Definition at line 68 of file config.h.

Referenced by kalmanInnovate().

#### 5.7.2.44 #define R1 1000.0f

R Matrix Diagonal Element 1.

Definition at line 71 of file config.h.

Referenced by kalmanInnovate().

#### 5.7.2.45 #define R2 1000.0f

R Matrix Diagonal Element 2.

Definition at line 72 of file config.h.

Referenced by kalmanInnovate().

#### 5.7.2.46 #define RX_BUFFER_SIZE 128

UART Receive Buffer Size.

Definition at line 163 of file config.h.

Referenced by ISR(), serialGet(), and serialRxBufferFull().

#### 5.7.2.47 #define SET_PITCHMINUS 2

Third Motor at the Bottom.

Definition at line 81 of file config.h.

#### 5.7.2.48 #define SET_PITCHPLUS 0

First Motor at the Top.

Definition at line 80 of file config.h.

#### 5.7.2.49 #define SET_ROLLMINUS 3

Fourth Motor at the Left.

Definition at line 79 of file config.h.

Referenced by setMotorSpeeds().

#### 5.7.2.50 #define SET_ROLLPLUS 1

Second Motor at the Right.

Definition at line 78 of file config.h.

Referenced by setMotorSpeeds().

#### 5.7.2.51 #define SOFTWARELOWPASS 50

Software Low-Pass on Gyro and ACC.

Definition at line 46 of file config.h.

#### 5.7.2.52 #define SPISS PB0

SPI Slave Select Pin.

Definition at line 151 of file config.h.

#### 5.7.2.53 #define TX_BUFFER_SIZE 128

UART Transmit Buffer Size.

Definition at line 164 of file config.h.

Referenced by ISR(), serialTxBufferFull(), and serialWrite().

#### 5.7.2.54 #define UART BLUETOOTH

UART module to use (0 to 3)

Definition at line 168 of file config.h.

#### 5.7.2.55 #define USB 0

FT232RL USB UART module.

Definition at line 166 of file config.h.

## 5.8 Debug Output

Allows debug ouput and assert usage.

### Files

- file debug.h

  *Debug and Assert Header and Implementation.*

### Macros

- #define DEBUGOUT(x) printf(x)

  *Debug Output Function.*
- #define ASSERTFUNC(x)

  *Simple Assert Implementation.*
- #define assert(x) ASSERTFUNC(x)

  *Enable assert()*
- #define debugPrint(ignore)

  *Disable debugPrint()*

### 5.8.1 Detailed Description

Allows debug ouput and assert usage. Usage: Before including this file, define DEBUG as the debuglevel, eg:

```
#define DEBUG 1
```

for debuglevel 1. Then use debugPrint("Foo") in your code. If you need to calculate stuff for your debug output, enclose it:

```
#if DEBUG >= 1
    debugPrint("Bar");
#endif
```

### 5.8.2 Macro Definition Documentation

#### 5.8.2.1 #define assert( *x* ) ASSERTFUNC(x)

Enable assert()

Definition at line 88 of file debug.h.

#### 5.8.2.2 #define ASSERTFUNC( *x* )

Simple Assert Implementation.

Definition at line 67 of file debug.h.

#### 5.8.2.3 #define DEBUGOUT( *x* ) printf(x)

Debug Output Function.

Definition at line 64 of file debug.h.

#### 5.8.2.4 #define debugPrint( *ignore* )

Disable debugPrint()

Definition at line 96 of file debug.h.

## 5.9 Error Reporting

Error reporting with human readable strings.

### Files

- file error.h

  *Global listing of different error conditions.*

### Macros

- #define CHECKERROR(x) if(x!=SUCCESS){return x;}

  *Check an Error Code.*
- #define REPORTERROR(x)

  *Report an error, if it occured.*

### Enumerations

- enum Error {
  SUCCESS = 0, TWI_NO_ANSWER, TWI_WRITE_ERROR, MALLOC_FAIL,
  ERROR, ARGUMENT_ERROR }

  *Error Conditions.*

### Functions

- char * getErrorString (Error e)

  *Returns a human-readable error description.*

### 5.9.1 Detailed Description

Error reporting with human readable strings.

### 5.9.2 Macro Definition Documentation

#### 5.9.2.1 #define CHECKERROR( *x* ) if(x!=SUCCESS){return x;}

Check an Error Code.

Return it if an error occured.

Definition at line 56 of file error.h.

Referenced by orientationInit(), and orientationTask().

#### 5.9.2.2 #define REPORTERROR( *x* )

**Value:**

```
{ \
    if (x != SUCCESS) { \
        char *s = getErrorString(x); \
        printf("Error: %s\n", s); \
        free(s); \
    } \
}
```

Report an error, if it occured.

Using printf()

**Examples:**

uartFlight.c.

Definition at line 59 of file error.h.

### 5.9.3 Enumeration Type Documentation

#### 5.9.3.1 enum Error

Error Conditions.

**Enumerator**

***SUCCESS*** No Error.

***TWI_NO_ANSWER*** No answer from TWI Slave.

***TWI_WRITE_ERROR*** Error while writing to TWI Slave.

***MALLOC_FAIL*** Malloc failed.

***ERROR*** General Error.

***ARGUMENT_ERROR*** Invalid arguments.

Definition at line 46 of file error.h.

```
46             {
47     SUCCESS = 0,
48     TWI_NO_ANSWER,
49     TWI_WRITE_ERROR,
50     MALLOC_FAIL,
51     ERROR,
52     ARGUMENT_ERROR,
53 } Error;
```

### 5.9.4 Function Documentation

#### 5.9.4.1 char* getErrorString ( Error *e* )

Returns a human-readable error description.

Free the string after use!

Definition at line 58 of file error.c.

References errorTable.

```
58                                {
59     char *buff = (char *)malloc(strlen_P((PGM_P)pgm_read_word(&(errorTable[e]))));
60     if (buff == NULL) {
61         return NULL;
62     }
63     strcpy_P(buff, (PGM_P)pgm_read_word(&(errorTable[e])));
64     return buff;
65 }
```

## 5.10 Gyroscope Driver

Configuring and reading an L3GD20.

### Files

- file gyro.h
  *L3GD20 Gyroscope API Header.*
- file gyro.c
  *L3GD20 Gyroscope API Implementation.*

### Macros

- #define GYROREG_CTRL1 0x20
  *Gyroscope Control Register 1.*
- #define GYROREG_CTRL4 0x23
  *Gyroscope Control Register 4.*
- #define GYROREG_OUTXL 0x28
  *First Gyroscope Output Register.*

### Enumerations

- enum GyroRange { r250DPS, r500DPS, r2000DPS }
  *Gyroscope Range options.*

### Functions

- Error gyroInit (GyroRange r)
  *Initializes the Gyroscope.*
- Error gyroRead (Vector3f *v)
  *Get a set of gyroscope data.*
- Error gyroWriteByte (uint8_t reg, uint8_t val)
  *Write a Gyroscope Register.*

### Variables

- GyroRange gyroRange
  *Stored range to scale returned values.*

### 5.10.1 Detailed Description

Configuring and reading an L3GD20.

### 5.10.2 Macro Definition Documentation

#### 5.10.2.1 #define GYROREG_CTRL1 0x20

Gyroscope Control Register 1.

Definition at line 48 of file gyro.c.

Referenced by gyroInit().

#### 5.10.2.2 #define GYROREG_CTRL4 0x23

Gyroscope Control Register 4.

Definition at line 49 of file gyro.c.

Referenced by gyroInit().

#### 5.10.2.3 #define GYROREG_OUTXL 0x28

First Gyroscope Output Register.

Definition at line 50 of file gyro.c.

Referenced by gyroRead().

### 5.10.3 Enumeration Type Documentation

#### 5.10.3.1 enum GyroRange

Gyroscope Range options.

**Enumerator**

***r250DPS*** +- 250 Degrees per Second

***r500DPS*** +- 500 Degrees per Second

***r2000DPS*** +- 2000 Degrees per Second

Definition at line 47 of file gyro.h.

```
47              {
48      r250DPS,
49      r500DPS,
50      r2000DPS,
51  } GyroRange;
```

### 5.10.4 Function Documentation

#### 5.10.4.1 Error gyroInit ( GyroRange *r* )

Initializes the Gyroscope.

I2C should already be initialized.

**Parameters**

| | |
|---|---|
| *r* | GyroRange to use |

**Returns**

TWI_NO_ANSWER, TWI_WRITE_ERROR, ARGUMENT_ERROR or SUCCESS

Definition at line 75 of file gyro.c.

References ARGUMENT_ERROR, gyroRange, GYROREG_CTRL1, GYROREG_CTRL4, gyroWriteByte(), r2000DPS, r250DPS, r500DPS, and SUCCESS.

Referenced by orientationInit().

```
75                              {
```

```
76      uint8_t v;
77      switch (r) {
78          case r250DPS:
79              v = 0x00;
80              break;
81          case r500DPS:
82              v = 0x10;
83              break;
84          case r2000DPS:
85              v = 0x20;
86              break;
87          default:
88              return ARGUMENT_ERROR;
89      }
90      gyroRange = r;
91      Error e = gyroWriteByte(GYROREG_CTRL1, 0x0F);
92      if (e != SUCCESS) {
93          return e;
94      }
95      e = gyroWriteByte(GYROREG_CTRL4, v);
96      return e;
97  }
```

#### 5.10.4.2 Error gyroRead ( Vector3f * v )

Get a set of gyroscope data.

gyroInit() should already be called.

**Parameters**

| | |
|---|---|
| *v* | Data Destionation |

**Returns**

TWI_NO_ANSWER, TWI_WRITE_ERROR, ARGUMENT_ERROR or SUCCESS

Definition at line 99 of file gyro.c.

References ARGUMENT_ERROR, GYRO_ADDRESS, GYROFILTERFACTOR, gyroRange, GYROREG_OUTXL, r2000DPS, r250DPS, r500DPS, SUCCESS, TWI_NO_ANSWER, TWI_READ, TWI_WRITE, TWI_WRITE_ERROR, twiReadAck(), twiReadNak(), twiRepStart(), twiStart(), twiWrite(), Vector3f::x, Vector3f::y, and Vector3f::z.

Referenced by orientationTask().

```
99                              {
100     // Simple Software Low-Pass
101     static double gyroSumX = 0, gyroSumY = 0, gyroSumZ = 0;
102     static double gyroFilterX = 0, gyroFilterY = 0, gyroFilterZ = 0;
103
104     if (v == NULL) {
105         return ARGUMENT_ERROR;
106     }
107     if (twiStart(GYRO_ADDRESS | TWI_WRITE)) {
108         return TWI_NO_ANSWER;
109     }
110     if (twiWrite(GYROREG_OUTXL | 0x80)) { // Auto Increment
111         return TWI_WRITE_ERROR;
112     }
113     if (twiRepStart(GYRO_ADDRESS | TWI_READ)) {
114         return TWI_NO_ANSWER;
115     }
116
117     uint8_t xl = twiReadAck();
118     uint8_t xh = twiReadAck();
119     uint8_t yl = twiReadAck();
120     uint8_t yh = twiReadAck();
121     uint8_t zl = twiReadAck();
122     uint8_t zh = twiReadNak();
123
124     int16_t x = *(int8_t *)(&xh);
125     x *= (1 << 8);
126     x |= xl;
127
128     int16_t y = *(int8_t *)(&yh);
```

```
129     y *= (1 << 8);
130     y |= yl;
131
132     int16_t z = *(int8_t *)(&zh);
133     z *= (1 << 8);
134     z |= zl;
135
136     switch (gyroRange) {
137         case r250DPS:
138             v->x = (((double)x) * 250 / 0x8000);
139             v->y = (((double)y) * 250 / 0x8000);
140             v->z = (((double)z) * 250 / 0x8000);
141             break;
142         case r500DPS:
143             v->x = (((double)x) * 500 / 0x8000);
144             v->y = (((double)y) * 500 / 0x8000);
145             v->z = (((double)z) * 500 / 0x8000);
146             break;
147         case r2000DPS:
148             v->x = (((double)x) * 2000 / 0x8000);
149             v->y = (((double)y) * 2000 / 0x8000);
150             v->z = (((double)z) * 2000 / 0x8000);
151             break;
152         default:
153             return ARGUMENT_ERROR;
154     }
155
156     gyroSumX = gyroSumX - gyroFilterX + v->x;
157     gyroFilterX = gyroSumX / GYROFILTERFACTOR;
158     v->x = gyroFilterX;
159
160     gyroSumY = gyroSumY - gyroFilterY + v->y;
161     gyroFilterY = gyroSumY / GYROFILTERFACTOR;
162     v->y = gyroFilterY;
163
164     gyroSumZ = gyroSumZ - gyroFilterZ + v->z;
165     gyroFilterZ = gyroSumZ / GYROFILTERFACTOR;
166     v->z = gyroFilterZ;
167
168     return SUCCESS;
169 }
```

#### 5.10.4.3 Error gyroWriteByte ( uint8_t *reg*, uint8_t *val* )

Write a Gyroscope Register.

I2C should aready be initialized!

**Parameters**

| | |
|---|---|
| *reg* | Register Address |
| *val* | New Value |

**Returns**

TWI_NO_ANSWER, TWI_WRITE_ERROR or SUCCESS.

Definition at line 61 of file gyro.c.

References TWI_NO_ANSWER.

Referenced by gyroInit().

```
61                                                  {
62     if (twiStart(GYRO_ADDRESS | TWI_WRITE)) {
63         return TWI_NO_ANSWER;
64     }
65     if (twiWrite(reg)) {
66         return TWI_WRITE_ERROR;
67     }
68     if (twiWrite(val)) {
69         return TWI_WRITE_ERROR;
70     }
71     twiStop();
72     return SUCCESS;
73 }
```

### 5.10.5 Variable Documentation

#### 5.10.5.1 GyroRange gyroRange

Stored range to scale returned values.

Definition at line 52 of file gyro.c.

Referenced by gyroInit(), and gyroRead().

## 5.11 Kalman-Filter

Kalman-Filter from Linus Helgesson

### Files

- file kalman.h

  *Kalman-Filter Header.*
- file kalman.c

  *Kalman-Filter Implementation.*

### Data Structures

- struct Kalman

  *Kalman-Filter State data.*

### Functions

- void kalmanInnovate (Kalman *data, double z1, double z2)

  *Step the Kalman Filter.*
- void kalmanInit (Kalman *data)

  *Initialize a Kalman-State.*

### 5.11.1 Detailed Description

Kalman-Filter from Linus Helgesson

### 5.11.2 Function Documentation

#### 5.11.2.1 void kalmanInit ( Kalman * *data* )

Initialize a Kalman-State.

**Parameters**

| | |
|---|---|
| *data* | Kalman-State to be initialized |

Definition at line 48 of file kalman.c.

References Kalman::p33, and Kalman::x3.

Referenced by orientationInit().

```
48                                {
49     data->x1 = 0.0f;
50     data->x2 = 0.0f;
51     data->x3 = 0.0f;
52
53     // Init P to diagonal matrix with large values since
54     // the initial state is not known
55     data->p11 = 1000.0f;
56     data->p12 = 0.0f;
57     data->p13 = 0.0f;
58     data->p21 = 0.0f;
59     data->p22 = 1000.0f;
60     data->p23 = 0.0f;
61     data->p31 = 0.0f;
62     data->p32 = 0.0f;
63     data->p33 = 1000.0f;
```

```
64 }
```

#### 5.11.2.2 void kalmanInnovate ( Kalman * *data*, double *z1*, double *z2* )

Step the Kalman Filter.

**Parameters**

| | |
|---|---|
| *data* | Kalman-Filter State |
| *z1* | Angle from Accelerometer |
| *z2* | Corresponding Gyroscope data |

Definition at line 66 of file kalman.c.

References DT, Kalman::p33, Q1, Q2, Q3, R1, R2, and Kalman::x3.

Referenced by orientationTask().

```
66                                                          {
67     double y1, y2;
68     double a, b, c;
69     double sDet;
70     double s11, s12, s21, s22;
71     double k11, k12, k21, k22, k31, k32;
72     double p11, p12, p13, p21, p22, p23, p31, p32, p33;
73 
74     // Step 1
75     // x(k) = Fx(k-1) + Bu + w:
76     data->x1 = data->x1 + DT*data->x2 - DT*data->x3;
77     //x2 = x2;
78     //x3 = x3;
79 
80     // Step 2
81     // P = FPF'+Q
82     a = data->p11 + data->p21*DT - data->p31*DT;
83     b = data->p12 + data->p22*DT - data->p32*DT;
84     c = data->p13 + data->p23*DT - data->p33*DT;
85     data->p11 = a + b*DT - c*DT + Q1;
86     data->p12 = b;
87     data->p13 = c;
88     data->p21 = data->p21 + data->p22*DT - data->p23*DT;
89     data->p22 = data->p22 + Q2;
90     //p23 = p23;
91     data->p31 = data->p31 + data->p32*DT - data->p33*DT;
92     //p32 = p32;
93     data->p33 = data->p33 + Q3;
94 
95     // Step 3
96     // y = z(k) - Hx(k)
97     y1 = z1-data->x1;
98     y2 = z2-data->x2;
99 
100     // Step 4
101     // S = HPT' + R
102     s11 = data->p11 + R1;
103     s12 = data->p12;
104     s21 = data->p21;
105     s22 = data->p22 + R2;
106 
107     // Step 5
108     // K = PH*inv(S)
109     sDet = 1/(s11*s22 - s12*s21);
110     k11 = (data->p11*s22 - data->p12*s21)*sDet;
111     k12 = (data->p12*s11 - data->p11*s12)*sDet;
112     k21 = (data->p21*s22 - data->p22*s21)*sDet;
113     k22 = (data->p22*s11 - data->p21*s12)*sDet;
114     k31 = (data->p31*s22 - data->p32*s21)*sDet;
115     k32 = (data->p32*s11 - data->p31*s12)*sDet;
116 
117     // Step 6
118     // x = x + Ky
119     data->x1 = data->x1 + k11*y1 + k12*y2;
120     data->x2 = data->x2 + k21*y1 + k22*y2;
121     data->x3 = data->x3 + k31*y1 + k32*y2;
122 
123     // Step 7
124     // P = (I-KH)P
125     p11 = data->p11*(1.0f - k11) - data->p21*k12;
```

```
126     p12 = data->p12*(1.0f - k11) - data->p22*k12;
127     p13 = data->p13*(1.0f - k11) - data->p23*k12;
128     p21 = data->p21*(1.0f - k22) - data->p11*k21;
129     p22 = data->p22*(1.0f - k22) - data->p12*k21;
130     p23 = data->p23*(1.0f - k22) - data->p13*k21;
131     p31 = data->p31 - data->p21*k32 - data->p11*k31;
132     p32 = data->p32 - data->p22*k32 - data->p12*k31;
133     p33 = data->p33 - data->p22*k32 - data->p13*k31;
134     data->p11 = p11; data->p12 = p12; data->p13 = p13;
135     data->p21 = p21; data->p22 = p22; data->p23 = p23;
136     data->p31 = p31; data->p32 = p32; data->p33 = p33;
137 }
```

## 5.12 Magnetometer Driver

Configuring and reading an LSM303DLHC Magnetometer.

### Files

- file mag.h

  *LSM303DLHC Magnetometer API Header.*
- file mag.c

  *LSM303DLHC Magnetometer API Implementation.*

### Macros

- #define MAGREG_CRB 0x01

  *Magnetometer Gain Register.*
- #define MAGREG_MR 0x02

  *Magnetometer Mode Register.*
- #define MAGREG_XH 0x03

  *First Magnetometer Output Register.*

### Enumerations

- enum MagRange {
  r1g3 = 1, r1g9 = 2, r2g5 = 3, r4g0 = 4,
  r4g7 = 5, r5g6 = 6, r8g1 = 7 }

  *Magnetometer Range options.*

### Functions

- Error magInit (MagRange r)

  *Initialize the Magnetometer.*
- Error magRead (Vector3f *v)

  *Read from the Magnetometer.*
- Error magWriteRegister (uint8_t reg, uint8_t val)

  *Write a Magnetometer Register.*

### 5.12.1 Detailed Description

Configuring and reading an LSM303DLHC Magnetometer.

### 5.12.2 Macro Definition Documentation

#### 5.12.2.1 #define MAGREG_CRB 0x01

Magnetometer Gain Register.

Definition at line 48 of file mag.c.

Referenced by magInit().

#### 5.12.2.2 #define MAGREG_MR 0x02

Magnetometer Mode Register.

Definition at line 49 of file mag.c.

Referenced by magInit().

#### 5.12.2.3 #define MAGREG_XH 0x03

First Magnetometer Output Register.

Definition at line 50 of file mag.c.

Referenced by magRead().

### 5.12.3 Enumeration Type Documentation

#### 5.12.3.1 enum MagRange

Magnetometer Range options.

**Enumerator**

***r1g3*** +- 1.3 Gauss

***r1g9*** +- 1.9 Gauss

***r2g5*** +- 2.5 Gauss

***r4g0*** +- 4.0 Gauss

***r4g7*** +- 4.7 Gauss

***r5g6*** +- 5.6 Gauss

***r8g1*** +- 8.1 Gauss

Definition at line 47 of file mag.h.

```
47              {
48     r1g3 = 1,
49     r1g9 = 2,
50     r2g5 = 3,
51     r4g0 = 4,
52     r4g7 = 5,
53     r5g6 = 6,
54     r8g1 = 7,
55 } MagRange;
```

### 5.12.4 Function Documentation

#### 5.12.4.1 Error magInit ( MagRange *r* )

Initialize the Magnetometer.

Call before magRead(). I2C should already be initialized!

**Parameters**

| | |
|---|---|
| *r* | MagRange to use. |

**Returns**

TWI_NO_ANSWER, TWI_WRITE_ERROR, ARGUMENT_ERROR or SUCCESS.

Definition at line 73 of file mag.c.

References ARGUMENT_ERROR, MAGREG_CRB, MAGREG_MR, magWriteRegister(), and SUCCESS.

Referenced by orientationInit().

```
73                              {
74     if ((r <= 0) || (r >= 8)) {
75         return ARGUMENT_ERROR;
76     }
77     Error e = magWriteRegister(MAGREG_MR, 0x00); // Continuous Conversion
78     if (e != SUCCESS) {
79         return e;
80     }
81     e = magWriteRegister(MAGREG_CRB, (r << 5)); // Set Range
82     return e;
83 }
```

#### 5.12.4.2 Error magRead ( Vector3f * *v* )

Read from the Magnetometer.

Magnetometer should already be initialized!

**Parameters**

| | |
|---|---|
| *v* | Vector3f for the read values |

**Returns**

TWI_NO_ANSWER, TWI_WRITE_ERROR, ARGUMENT_ERROR or SUCCESS.

Definition at line 85 of file mag.c.

References ARGUMENT_ERROR, MAG_ADDRESS, MAGREG_XH, SUCCESS, TWI_NO_ANSWER, TWI_READ, TWI_WRITE, TWI_WRITE_ERROR, twiReadAck(), twiReadNak(), twiRepStart(), twiStart(), twiWrite(), Vector3f::x, Vector3f::y, and Vector3f::z.

```
85                                {
86     if (v == NULL) {
87         return ARGUMENT_ERROR;
88     }
89     if (twiStart(MAG_ADDRESS | TWI_WRITE)) {
90         return TWI_NO_ANSWER;
91     }
92     if (twiWrite(MAGREG_XH)) {
93         return TWI_WRITE_ERROR;
94     }
95     if (twiRepStart(MAG_ADDRESS | TWI_READ)) {
96         return TWI_NO_ANSWER;
97     }
98     uint8_t xh = twiReadAck();
99     uint8_t xl = twiReadAck();
100     uint8_t zh = twiReadAck();
101     uint8_t zl = twiReadAck();
102     uint8_t yh = twiReadAck();
103     uint8_t yl = twiReadNak();
104
105     v->x = (int16_t)(xh << 8 | xl);
106     v->y = (int16_t)(yh << 8 | yl);
107     v->z = (int16_t)(zh << 8 | zl);
108     return SUCCESS;
109 }
```

#### 5.12.4.3 Error magWriteRegister ( uint8_t *reg*, uint8_t *val* )

Write a Magnetometer Register.

I2C should aready be initialized!

**Parameters**

| | |
|---|---|
| *reg* | Register Address |
| *val* | New Value |

**Returns**

TWI_NO_ANSWER, TWI_WRITE_ERROR or SUCCESS.

Definition at line 59 of file mag.c.

References TWI_NO_ANSWER.

Referenced by magInit().

```
59                                                  {
60     if (twiStart(MAG_ADDRESS | TWI_WRITE)) {
61         return TWI_NO_ANSWER;
62     }
63     if (twiWrite(reg)) {
64         return TWI_WRITE_ERROR;
65     }
66     if (twiWrite(val)) {
67         return TWI_WRITE_ERROR;
68     }
69     twiStop();
70     return SUCCESS;
71 }
```

## 5.13 Motor Controller Driver

Controlling four BL-Ctrl V1.2 Brushless controllers.

### Files

- file motor.h

  *BL-Ctrl V1.2 Controller API Header.*
- file motor.c

  *BL-Ctrl V1.2 Controller API Implementation.*

### Functions

- void motorInit (void)

  *Initializes the motor control library.*
- void motorSet (uint8_t id, uint8_t speed)

  *Set the speed of one or all motors.*
- void motorTask (void)

  *Send the values stored in motorSpeed to the Controllers.*

### Variables

- uint8_t motorSpeed [MOTORCOUNT]

  *Speed for the four motors.*
- uint8_t motorSpeed [MOTORCOUNT]

  *Speed for the four motors.*

### 5.13.1 Detailed Description

Controlling four BL-Ctrl V1.2 Brushless controllers.

### 5.13.2 Function Documentation

#### 5.13.2.1 void motorInit ( void )

Initializes the motor control library.

Really only sets motorSpeed to zero.

**Examples:**

uartFlight.c.

Definition at line 58 of file motor.c.

References MOTORCOUNT, and motorSpeed.

```
58                      {
59      for (uint8_t i = 0; i < MOTORCOUNT; i++) {
60          motorSpeed[i] = 0;
61      }
62  }
```

#### 5.13.2.2 void motorSet ( uint8_t *id*, uint8_t *speed* )

Set the speed of one or all motors.

**Parameters**

| | |
|---|---|
| *id* | Motor ID (0 to 3, 4 = all) |
| *speed* | New Speed |

Definition at line 64 of file motor.c.

References MOTORCOUNT, and motorSpeed.

Referenced by setMotorSpeeds().

```
64                                              {
65     if (id < MOTORCOUNT) {
66         motorSpeed[id] = speed;
67     } else {
68         for (id = 0; id < MOTORCOUNT; id++) {
69             motorSpeed[id] = speed;
70         }
71     }
72 }
```

#### 5.13.2.3 void motorTask ( void )

Send the values stored in motorSpeed to the Controllers.

I2C already has to be initialized!

**Examples:**

uartFlight.c.

Definition at line 50 of file motor.c.

References MOTOR_BASEADDRESS, MOTORCOUNT, motorSpeed, TWI_WRITE, twiStart(), twiStop(), and twiWrite().

```
50                      {
51     for (uint8_t i = 0; i < MOTORCOUNT; i++) {
52         twiStart(MOTOR_BASEADDRESS + (i << 1) +
       TWI_WRITE);
53         twiWrite(motorSpeed[i]);
54         twiStop();
55     }
56 }
```

### 5.13.3 Variable Documentation

#### 5.13.3.1 uint8_t motorSpeed[MOTORCOUNT]

Speed for the four motors.

**Examples:**

uartFlight.c.

Definition at line 48 of file motor.c.

Referenced by motorInit(), motorSet(), and motorTask().

#### 5.13.3.2 uint8_t motorSpeed[MOTORCOUNT]

Speed for the four motors.

Definition at line 48 of file motor.c.

Referenced by motorInit(), motorSet(), and motorTask().

## 5.14 Orientation Calculation

Calculate Orientation using the Kalman-Filter, Accelerometer and Gyroscope.

### Files

- file orientation.h

  *Orientation API Header.*
- file orientation.c

  *Orientation API Implementation.*

### Data Structures

- struct Angles

  *Can store orientation in Euler Space.*

### Macros

- #define TODEG(x) ((x ∗ 180) / M_PI)

  *Convert Radians to Degrees.*

### Functions

- Error orientationInit (void)

  *Initializes the Orientation API.*
- Error orientationTask (void)

  *Calculate the current orientation.*
- void zeroOrientation (void)

  *Sets the current orientation to zero.*

### Variables

- Angles orientation

  *Current Aircraft orientation.*
- Angles orientation = {.pitch = 0, .roll = 0, .yaw = 0}

  *Current Aircraft orientation.*
- Angles orientationError = {.pitch = 0, .roll = 0, .yaw = 0}

  *Current Aircraft orientation offset.*
- Kalman pitchData

  *Kalman-State for Pitch Angle.*
- Kalman rollData

  *Kalman-State for Roll Angle.*

### 5.14.1 Detailed Description

Calculate Orientation using the Kalman-Filter, Accelerometer and Gyroscope.

### 5.14.2 Macro Definition Documentation

#### 5.14.2.1 #define TODEG( x ) ((x * 180) / M_PI)

Convert Radians to Degrees.

Definition at line 54 of file orientation.c.

Referenced by orientationTask().

### 5.14.3 Function Documentation

#### 5.14.3.1 Error orientationInit ( void )

Initializes the Orientation API.

Also initializes the Accelerometer, Gyroscope and Magnetometer. I2C should already be initialized!

**Returns**

TWI_NO_ANSWER, TWI_WRITE_ERROR, ARGUMENT_ERROR or SUCCESS.

**Examples:**

uartFlight.c.

Definition at line 65 of file orientation.c.

References accInit(), CHECKERROR, gyroInit(), kalmanInit(), magInit(), r1g9, r250DPS, r4G, and SUCCESS.

```
65                          {
66     Error e = accInit(r4G);
67     CHECKERROR(e);
68     e = gyroInit(r250DPS);
69     CHECKERROR(e);
70     e = magInit(r1g9);
71     CHECKERROR(e);
72     kalmanInit(&pitchData);
73     kalmanInit(&rollData);
74     return SUCCESS;
75 }
```

#### 5.14.3.2 Error orientationTask ( void )

Calculate the current orientation.

It will be stored in the global orientation Struct.

**Returns**

TWI_NO_ANSWER, TWI_WRITE_ERROR, ARGUMENT_ERROR or SUCCESS.

**Examples:**

uartFlight.c.

Definition at line 77 of file orientation.c.

References accRead(), CHECKERROR, gyroRead(), kalmanInnovate(), orientation, Angles::pitch, Angles::roll, SUCCESS, TODEG, Vector3f::x, Vector3f::y, and Vector3f::z.

```
77                          {
78     Vector3f g, a;
79     Error e = accRead(&a); // Read Accelerometer
```

```
80     CHECKERROR(e);
81     e = gyroRead(&g); // Read Gyroscope
82     CHECKERROR(e);
83
84     // Calculate Pitch & Roll from Accelerometer Data
85     double roll = atan(a.x / hypot(a.y, a.z));
86     double pitch = atan(a.y / hypot(a.x, a.z));
87     roll = TODEG(roll);
88     pitch = TODEG(pitch); // As Degree, not radians!
89
90     // Filter Roll and Pitch with Gyroscope Data from the corresponding axis
91     kalmanInnovate(&pitchData, pitch, g.x);
92     kalmanInnovate(&rollData, roll, g.y);
93     orientation.roll = rollData.x1 - orientationError.
       roll;
94     orientation.pitch = pitchData.x1 - orientationError.
       pitch;
95
96     //orientation.roll = round(orientation.roll * 10) / 10;
97     //orientation.pitch = round(orientation.pitch * 10) / 10;
98
99     return SUCCESS;
100 }
```

#### 5.14.3.3 void zeroOrientation ( void )

Sets the current orientation to zero.

**Examples:**

uartFlight.c.

Definition at line 102 of file orientation.c.

References orientation, Angles::pitch, Angles::roll, and Angles::yaw.

```
102                          {
103     orientationError.roll = orientation.roll +
       orientationError.roll;
104     orientationError.pitch = orientation.pitch +
       orientationError.pitch;
105     orientationError.yaw = orientation.yaw +
       orientationError.yaw;
106 }
```

### 5.14.4 Variable Documentation

#### 5.14.4.1 Angles orientation

Current Aircraft orientation.

**Examples:**

uartFlight.c.

Definition at line 57 of file orientation.c.

Referenced by orientationTask(), pidTask(), and zeroOrientation().

#### 5.14.4.2 Angles orientation = {.pitch = 0, .roll = 0, .yaw = 0}

Current Aircraft orientation.

Definition at line 57 of file orientation.c.

Referenced by orientationTask(), pidTask(), and zeroOrientation().

#### 5.14.4.3 Angles orientationError = {.pitch = 0, .roll = 0, .yaw = 0}

Current Aircraft orientation offset.

Definition at line 60 of file orientation.c.

#### 5.14.4.4 Kalman pitchData

Kalman-State for Pitch Angle.

Definition at line 62 of file orientation.c.

#### 5.14.4.5 Kalman rollData

Kalman-State for Roll Angle.

Definition at line 63 of file orientation.c.

## 5.15 PID-Controller

Simple implementation for multiple floating-point PID Controllers.

### Files

- file pid.h

  *PID Library Header.*
- file pid.c

  *PID Library Implementation.*

### Data Structures

- struct PIDState

  *Data Structure for a single PID Controller.*

### Macros

- #define ROLL 0

  *Roll index for o_should, o_output and o_pids.*
- #define PITCH 1

  *Pitch index for o_should, o_output and o_pids.*

### Functions

- void pidInit (void)

  *Initialize Roll and Pitch PID.*
- void pidTask (void)

  *Step the Roll and Pitch PID Controllers.*
- void pidSet (PIDState *pid, double kp, double ki, double kd, double min, double max, double iMin, double iMax)

  *Set the parameters of a PID controller.*
- double pidExecute (double should, double is, PIDState *state)

  *Execute a single PID Control Step.*

### Variables

- double o_should [2]

  *Roll and Pitch target angles.*
- double o_output [2]

  *Roll and Pitch PID Output.*
- PIDState o_pids [2]

  *Roll and Pitch PID States.*
- PIDState o_pids [2]

  *Roll and Pitch PID States.*
- double o_should [2]

  *Roll and Pitch target angles.*
- double o_output [2]

  *Roll and Pitch PID Output.*

### 5.15.1 Detailed Description

Simple implementation for multiple floating-point PID Controllers.

### 5.15.2 Macro Definition Documentation

#### 5.15.2.1 #define PITCH 1

Pitch index for o_should, o_output and o_pids.

**Examples:**

uartFlight.c.

Definition at line 61 of file pid.h.

Referenced by pidTask().

#### 5.15.2.2 #define ROLL 0

Roll index for o_should, o_output and o_pids.

**Examples:**

uartFlight.c.

Definition at line 60 of file pid.h.

Referenced by pidTask().

### 5.15.3 Function Documentation

#### 5.15.3.1 double pidExecute ( double *should*, double *is*, PIDState * *state* )

Execute a single PID Control Step.

**Parameters**

| | |
|---|---|
| *should* | Target value |
| *is* | Measured value |
| *state* | PID State |

**Returns**

PID Output

Definition at line 54 of file pid.c.

References getSystemTime(), PIDState::intMax, PIDState::intMin, PIDState::kd, PIDState::ki, PIDState::kp, PIDState::last, PIDState::lastError, PIDState::outMax, PIDState::outMin, and PIDState::sumError.

Referenced by pidTask().

```
54                                                      {
55     time_t now = getSystemTime();
56     double timeChange = (double)(now - state->last);
57     double error = should - is;
58     double newErrorSum = state->sumError + (error * timeChange);
59     if ((newErrorSum >= state->intMin) && (newErrorSum <= state->intMax))
```

```
60          state->sumError = newErrorSum; // Prevent Integral Windup
61      double dError = (error - state->lastError) / timeChange;
62      double output = (state->kp * error) + (state->ki * state->sumError) + (state->
        kd * dError);
63      state->lastError = error;
64      state->last = now;
65      if (output > state->outMax) {
66          output = state->outMax;
67      }
68      if (output < state->outMin) {
69          output = state->outMin;
70      }
71      return output;
72  }
```

#### 5.15.3.2 void pidInit ( void )

Initialize Roll and Pitch PID.

Stores the PID States in o_pids. Also resets o_should to zero.

**Examples:**

uartFlight.c.

Definition at line 74 of file pid.c.

References o_should, PID_D, PID_I, PID_INTMAX, PID_INTMIN, PID_OUTMAX, PID_OUTMIN, PID_P, and pidSet().

```
74                      {
75      for (uint8_t i = 0; i < 2; i++) {
76          pidSet(&o_pids[i], PID_P, PID_I, PID_D,
        PID_OUTMIN, PID_OUTMAX, PID_INTMIN, PID_INTMAX);
77          o_should[i] = 0.0;
78      }
79  }
```

#### 5.15.3.3 void pidSet ( PIDState * *pid*, double *kp*, double *ki*, double *kd*, double *min*, double *max*, double *iMin*, double *iMax* )

Set the parameters of a PID controller.

The state variables will be reset to zero.

**Parameters**

| | |
|---|---|
| *pid* | PIDState to be changed. |
| *kp* | New Proportional constant. |
| *ki* | New Integral constant. |
| *kd* | New Derivative constant. |
| *min* | New minimum Output. |
| *max* | New maximum Output. |
| *iMin* | New minimal Integral Sum. |
| *iMax* | New maximal Integral Sum. |

**Examples:**

uartFlight.c.

Definition at line 81 of file pid.c.

References PIDState::intMax, PIDState::intMin, PIDState::kd, PIDState::ki, PIDState::kp, PIDState::last, PIDState::lastError, PIDState::outMax, PIDState::outMin, and PIDState::sumError.

Referenced by pidInit().

```
81
          {
82      pid->kp = kp;
83      pid->ki = ki;
84      pid->kd = kd;
85      pid->outMin = min;
86      pid->outMax = max;
87      pid->intMin = iMin;
88      pid->intMax = iMax;
89      pid->lastError = 0;
90      pid->sumError = 0;
91      pid->last = 0;
92  }
```

#### 5.15.3.4 void pidTask ( void )

Step the Roll and Pitch PID Controllers.

Placing their output in o_output and reading the input from o_should and the global orientation Angles.

**Examples:**

uartFlight.c.

Definition at line 94 of file pid.c.

References o_output, o_should, orientation, pidExecute(), Angles::pitch, PITCH, Angles::roll, and ROLL.

```
94                      {
95      o_output[ROLL] = pidExecute(o_should[ROLL],
        orientation.roll, &o_pids[ROLL]);
96      o_output[PITCH] = pidExecute(o_should[PITCH],
        orientation.pitch, &o_pids[PITCH]);
97  }
```

### 5.15.4 Variable Documentation

#### 5.15.4.1 double o_output[2]

Roll and Pitch PID Output.

Definition at line 52 of file pid.c.

Referenced by pidTask(), and setTask().

#### 5.15.4.2 double o_output[2]

Roll and Pitch PID Output.

**Examples:**

uartFlight.c.

Definition at line 52 of file pid.c.

Referenced by pidTask(), and setTask().

#### 5.15.4.3 PIDState o_pids[2]

Roll and Pitch PID States.

Definition at line 50 of file pid.c.

Referenced by setTask().

#### 5.15.4.4 PIDState o_pids[2]

Roll and Pitch PID States.

**Examples:**

uartFlight.c.

Definition at line 50 of file pid.c.

Referenced by setTask().

#### 5.15.4.5 double o_should[2]

Roll and Pitch target angles.

Definition at line 51 of file pid.c.

Referenced by pidInit(), and pidTask().

#### 5.15.4.6 double o_should[2]

Roll and Pitch target angles.

**Examples:**

uartFlight.c.

Definition at line 51 of file pid.c.

Referenced by pidInit(), and pidTask().

## 5.16 UART Driver

Serial Library for AVRs built-in UART Hardware.

### Files

- file serial.h
  *UART API Header.*
- file serial.c
  *UART API Implementation.*

### Macros

- #define BAUD(baudRate, xtalCpu) ((xtalCpu)/((baudRate)∗16l)-1)
  *Calculate Baudrate Register Value.*
- #define XON 0x11
  *XON Byte.*
- #define XOFF 0x13
  *XOFF Byte.*
- #define FLOWMARK 5
  *Space remaining to trigger XOFF/XON.*

### Functions

- void serialInit (uint16_t baud)
  *Initialize the UART Hardware.*
- void serialClose (void)
  *Stop the UART Hardware.*
- void setFlow (uint8_t on)
  *Manually change the flow control.*
- uint8_t serialHasChar (void)
  *Check if a byte was received.*
- uint8_t serialGet (void)
  *Read a single byte.*
- uint8_t serialGetBlocking (void)
  *Wait until a character is received.*
- uint8_t serialRxBufferFull (void)
  *Check if the receive buffer is full.*
- uint8_t serialRxBufferEmpty (void)
  *Check if the receive buffer is empty.*
- void serialWrite (uint8_t data)
  *Send a byte.*
- void serialWriteString (const char ∗data)
  *Send a string.*
- uint8_t serialTxBufferFull (void)
  *Check if the transmit buffer is full.*
- uint8_t serialTxBufferEmpty (void)
  *Check if the transmit buffer is empty.*
- ISR (SERIALRECIEVEINTERRUPT)
  *Receive Complete Interrupt.*
- ISR (SERIALTRANSMITINTERRUPT)
  *Data Register Empty Interrupt.*

### Variables

- uint8_t volatile rxBuffer [RX_BUFFER_SIZE]

  *RX FIFO Buffer.*
- uint8_t volatile txBuffer [TX_BUFFER_SIZE]

  *TX FIFO Buffer.*
- uint16_t volatile rxRead = 0

  *RX FIFO Read Position.*
- uint16_t volatile rxWrite = 0

  *RX FIFO Write Position.*
- uint16_t volatile txRead = 0

  *TX FIFO Read Position.*
- uint16_t volatile txWrite = 0

  *TX FIFO Write Position.*
- uint8_t volatile shouldStartTransmission = 1

  *Should enable interrupt.*

## 5.16.1 Detailed Description

Serial Library for AVRs built-in UART Hardware. Allows XON/XOFF Flow Control. FIFO Buffer for Receiving and Transmitting.

## 5.16.2 Macro Definition Documentation

### 5.16.2.1 #define BAUD( *baudRate, xtalCpu* ) ((xtalCpu)/((baudRate)∗16l)-1)

Calculate Baudrate Register Value.

Definition at line 46 of file serial.h.

Referenced by xyInit().

### 5.16.2.2 #define FLOWMARK 5

Space remaining to trigger XOFF/XON.

Definition at line 63 of file serial.c.

Referenced by ISR(), and serialGet().

### 5.16.2.3 #define XOFF 0x13

XOFF Byte.

Definition at line 47 of file serial.c.

Referenced by ISR(), and setFlow().

### 5.16.2.4 #define XON 0x11

XON Byte.

Definition at line 46 of file serial.c.

Referenced by serialGet(), and setFlow().

### 5.16.3 Function Documentation

#### 5.16.3.1 ISR ( SERIALRECIEVEINTERRUPT )

Receive Complete Interrupt.

Definition at line 107 of file serial.c.

References FLOWMARK, RX_BUFFER_SIZE, rxBuffer, rxWrite, shouldStartTransmission, and XOFF.

```
107                             {
108     rxBuffer[rxWrite] = SERIALDATA;
109     if (rxWrite < (RX_BUFFER_SIZE - 1)) {
110         rxWrite++;
111     } else {
112         rxWrite = 0;
113     }
114
115 #ifdef FLOWCONTROL
116     rxBufferElements++;
117     if ((flow == 1) && (rxBufferElements >= (RX_BUFFER_SIZE -
    FLOWMARK))) {
118         sendThisNext = XOFF;
119         flow = 0;
120         if (shouldStartTransmission) {
121             shouldStartTransmission = 0;
122             SERIALB |= (1 << SERIALUDRIE);
123             SERIALA |= (1 << SERIALUDRE); // Trigger Interrupt
124         }
125     }
126 #endif
127 }
```

#### 5.16.3.2 ISR ( SERIALTRANSMITINTERRUPT )

Data Register Empty Interrupt.

Definition at line 130 of file serial.c.

References shouldStartTransmission, TX_BUFFER_SIZE, txBuffer, txRead, and txWrite.

```
130                             {
131 #ifdef FLOWCONTROL
132     if (sendThisNext) {
133         SERIALDATA = sendThisNext;
134         sendThisNext = 0;
135     } else {
136 #endif
137         if (txRead != txWrite) {
138             SERIALDATA = txBuffer[txRead];
139             if (txRead < (TX_BUFFER_SIZE -1)) {
140                 txRead++;
141             } else {
142                 txRead = 0;
143             }
144         } else {
145             shouldStartTransmission = 1;
146             SERIALB &= ~(1 << SERIALUDRIE); // Disable Interrupt
147         }
148 #ifdef FLOWCONTROL
149     }
150 #endif
151 }
```

#### 5.16.3.3 void serialClose ( void )

Stop the UART Hardware.

Definition at line 164 of file serial.c.

References rxRead, rxWrite, serialTxBufferEmpty(), shouldStartTransmission, txRead, and txWrite.

```
164                         {
165     uint8_t sreg = SREG;
166     sei();
167     while (!serialTxBufferEmpty());
168     while (SERIALB & (1 << SERIALUDRIE)); // Wait while Transmit Interrupt is on
169     cli();
170     SERIALB = 0;
171     SERIALC = 0;
172     rxRead = 0;
173     txRead = 0;
174     rxWrite = 0;
175     txWrite = 0;
176     shouldStartTransmission = 1;
177 #ifdef FLOWCONTROL
178     flow = 1;
179     sendThisNext = 0;
180     rxBufferElements = 0;
181 #endif
182     SREG = sreg;
183 }
```

#### 5.16.3.4 uint8_t serialGet ( void )

Read a single byte.

**Returns**

Received byte or 0

Definition at line 231 of file serial.c.

References FLOWMARK, RX_BUFFER_SIZE, rxBuffer, rxRead, rxWrite, shouldStartTransmission, and XON.

Referenced by serialGetBlocking(), uartinput(), and uartMenuTask().

```
231                         {
232     uint8_t c;
233
234 #ifdef FLOWCONTROL
235     rxBufferElements--;
236     if ((flow == 0) && (rxBufferElements <= FLOWMARK)) {
237         while (sendThisNext != 0);
238         sendThisNext = XON;
239         flow = 1;
240         if (shouldStartTransmission) {
241             shouldStartTransmission = 0;
242             SERIALB |= (1 << SERIALUDRIE);
243             SERIALA |= (1 << SERIALUDRE); // Trigger Interrupt
244         }
245     }
246 #endif
247
248     if (rxRead != rxWrite) {
249         c = rxBuffer[rxRead];
250         rxBuffer[rxRead] = 0;
251         if (rxRead < (RX_BUFFER_SIZE - 1)) {
252             rxRead++;
253         } else {
254             rxRead = 0;
255         }
256         return c;
257     } else {
258         return 0;
259     }
260 }
```

#### 5.16.3.5 uint8_t serialGetBlocking ( void )

Wait until a character is received.

**Returns**

Received byte

Definition at line 226 of file serial.c.

References serialGet(), and serialHasChar().

```
226                                 {
227     while(!serialHasChar());
228     return serialGet();
229 }
```

#### 5.16.3.6 uint8_t serialHasChar ( void )

Check if a byte was received.

**Returns**

1 if a byte was received, 0 if not

Definition at line 218 of file serial.c.

References rxRead, and rxWrite.

Referenced by serialGetBlocking(), uartinput(), and uartMenuTask().

```
218                                 {
219     if (rxRead != rxWrite) { // True if char available
220         return 1;
221     } else {
222         return 0;
223     }
224 }
```

#### 5.16.3.7 void serialInit ( uint16_t *baud* )

Initialize the UART Hardware.

**Parameters**

| | |
|---|---|
| *baud* | Baudrate. Use the BAUD() macro! |

Definition at line 153 of file serial.c.

Referenced by xyInit().

```
153                                 {
154     // Default: 8N1
155     SERIALC = (1 << SERIALUCSZ0) | (1 << SERIALUCSZ1);
156 
157     // Set baudrate
158     SERIALUBRR = baud;
159 
160     SERIALB = (1 << SERIALRXCIE); // Enable Interrupts
161     SERIALB |= (1 << SERIALRXEN) | (1 << SERIALTXEN); // Enable Receiver/Transmitter
162 }
```

#### 5.16.3.8 uint8_t serialRxBufferEmpty ( void )

Check if the receive buffer is empty.

**Returns**

1 if buffer is empty, 0 if not.

Definition at line 266 of file serial.c.

References rxRead, and rxWrite.

```
266                                 {
267     if (rxRead != rxWrite) {
268         return 0;
269     } else {
270         return 1;
271     }
272 }
```

#### 5.16.3.9 uint8_t serialRxBufferFull ( void )

Check if the receive buffer is full.

**Returns**

1 if buffer is full, 0 if not

Definition at line 262 of file serial.c.

References RX_BUFFER_SIZE, rxRead, and rxWrite.

```
262                                 {
263     return (((rxWrite + 1) == rxRead) || ((rxRead == 0) && ((
      rxWrite + 1) == RX_BUFFER_SIZE)));
264 }
```

#### 5.16.3.10 uint8_t serialTxBufferEmpty ( void )

Check if the transmit buffer is empty.

**Returns**

1 if buffer is empty, 0 if not.

Definition at line 313 of file serial.c.

References txRead, and txWrite.

Referenced by serialClose().

```
313                                 {
314     if (txRead != txWrite) {
315         return 0;
316     } else {
317         return 1;
318     }
319 }
```

#### 5.16.3.11 uint8_t serialTxBufferFull ( void )

Check if the transmit buffer is full.

**Returns**

1 if buffer is full, 0 if not

Definition at line 309 of file serial.c.

References TX_BUFFER_SIZE, txRead, and txWrite.

Referenced by serialWrite().

```
309                                 {
310     return (((txWrite + 1) == txRead) || ((txRead == 0) && ((
      txWrite + 1) == TX_BUFFER_SIZE)));
311 }
```

#### 5.16.3.12 void serialWrite ( uint8_t *data* )

Send a byte.

**Parameters**

| | |
|---|---|
| *data* | Byte to send |

Definition at line 278 of file serial.c.

References serialTxBufferFull(), shouldStartTransmission, TX_BUFFER_SIZE, txBuffer, and txWrite.

Referenced by serialWriteString(), and uartoutput().

```
278                                 {
279 #ifdef SERIALINJECTCR
280     if (data == '\n') {
281         serialWrite('\r');
282     }
283 #endif
284     while (serialTxBufferFull());
285 
286     txBuffer[txWrite] = data;
287     if (txWrite < (TX_BUFFER_SIZE - 1)) {
288         txWrite++;
289     } else {
290         txWrite = 0;
291     }
292     if (shouldStartTransmission) {
293         shouldStartTransmission = 0;
294         SERIALB |= (1 << SERIALUDRIE); // Enable Interrupt
295         SERIALA |= (1 << SERIALUDRE); // Trigger Interrupt
296     }
297 }
```

#### 5.16.3.13 void serialWriteString ( const char * *data* )

Send a string.

**Parameters**

| | |
|---|---|
| *data* | Null-Terminated String |

Definition at line 299 of file serial.c.

References serialWrite().

```
299                                         {
300     if (data == 0) {
301         serialWriteString("NULL");
302     } else {
303         while (*data != '\0') {
304             serialWrite(*data++);
```

```
305         }
306     }
307 }
```

#### 5.16.3.14 void setFlow ( uint8_t *on* )

Manually change the flow control.

**Parameters**

| | |
|---|---|
| *on* | 1 of on, 0 if off |

Definition at line 185 of file serial.c.

References shouldStartTransmission, XOFF, and XON.

```
185                         {
186 #ifdef FLOWCONTROL
187     if (flow != on) {
188         if (on == 1) {
189             // Send XON
190             while (sendThisNext != 0);
191             sendThisNext = XON;
192             flow = 1;
193             if (shouldStartTransmission) {
194                 shouldStartTransmission = 0;
195                 SERIALB |= (1 << SERIALUDRIE);
196                 SERIALA |= (1 << SERIALUDRE); // Trigger Interrupt
197             }
198         } else {
199             // Send XOFF
200             sendThisNext = XOFF;
201             flow = 0;
202             if (shouldStartTransmission) {
203                 shouldStartTransmission = 0;
204                 SERIALB |= (1 << SERIALUDRIE);
205                 SERIALA |= (1 << SERIALUDRE); // Trigger Interrupt
206             }
207         }
208         // Wait till it's transmitted
209         while (SERIALB & (1 << SERIALUDRIE));
210     }
211 #endif
212 }
```

### 5.16.4 Variable Documentation

#### 5.16.4.1 uint8_t volatile rxBuffer[RX_BUFFER_SIZE]

RX FIFO Buffer.

Definition at line 92 of file serial.c.

Referenced by ISR(), and serialGet().

#### 5.16.4.2 uint16_t volatile rxRead = 0

RX FIFO Read Position.

Definition at line 94 of file serial.c.

Referenced by serialClose(), serialGet(), serialHasChar(), serialRxBufferEmpty(), and serialRxBufferFull().

#### 5.16.4.3 uint16_t volatile rxWrite = 0

RX FIFO Write Position.

Definition at line 95 of file serial.c.

Referenced by ISR(), serialClose(), serialGet(), serialHasChar(), serialRxBufferEmpty(), and serialRxBufferFull().

#### 5.16.4.4 uint8_t volatile shouldStartTransmission = 1

Should enable interrupt.

Definition at line 98 of file serial.c.

Referenced by ISR(), serialClose(), serialGet(), serialWrite(), and setFlow().

#### 5.16.4.5 uint8_t volatile txBuffer[TX_BUFFER_SIZE]

TX FIFO Buffer.

Definition at line 93 of file serial.c.

Referenced by ISR(), and serialWrite().

#### 5.16.4.6 uint16_t volatile txRead = 0

TX FIFO Read Position.

Definition at line 96 of file serial.c.

Referenced by ISR(), serialClose(), serialTxBufferEmpty(), and serialTxBufferFull().

#### 5.16.4.7 uint16_t volatile txWrite = 0

TX FIFO Write Position.

Definition at line 97 of file serial.c.

Referenced by ISR(), serialClose(), serialTxBufferEmpty(), serialTxBufferFull(), and serialWrite().

## 5.17 Motor Speed Mixer

Takes the Base Speed and PID-Output and sets Motor Speed accordingly.

### Files

- file set.h

  *Motor Mixer Library Header.*
- file set.c

  *Motor Mixer Library Implementation.*

### Macros

- #define MAXDIFF (baseSpeed ∗ PID_FACTOR)

  *Maximum Speed difference on one axis.*

### Functions

- void setTask (void)

  *Read the PID Output and Set the Motor Speeds.*
- void setMotorSpeeds (uint8_t axis, uint8_t ∗vals)

  *Set the Motor Speeds according to the SET_∗ Motor Position Constants.*

### Variables

- uint8_t baseSpeed

  *Motor Base Speed.*
- uint8_t baseSpeed = 0

  *Motor Base Speed.*

### 5.17.1 Detailed Description

Takes the Base Speed and PID-Output and sets Motor Speed accordingly.

### 5.17.2 Macro Definition Documentation

#### 5.17.2.1 #define MAXDIFF (baseSpeed ∗ PID_FACTOR)

Maximum Speed difference on one axis.

Definition at line 51 of file set.c.

Referenced by setTask().

### 5.17.3 Function Documentation

#### 5.17.3.1 void setMotorSpeeds ( uint8_t *axis*, uint8_t ∗ *vals* ) [inline]

Set the Motor Speeds according to the SET_∗ Motor Position Constants.

**Parameters**

| | |
|---|---|
| *axis* | ROLL or PITCH |
| *vals* | Speeds for the two Motors on this axis (+, -) |

Definition at line 59 of file set.c.

References motorSet(), SET_ROLLMINUS, and SET_ROLLPLUS.

Referenced by setTask().

```
59                                                      {
60     if (axis == ROLL) {
61         motorSet(SET_ROLLPLUS, vals[0]);
62         motorSet(SET_ROLLMINUS, vals[1]);
63     } else if (axis == PITCH) {
64         motorSet(SET_PITCHPLUS, vals[0]);
65         motorSet(SET_PITCHMINUS, vals[1]);
66     }
67 }
```

#### 5.17.3.2 void setTask ( void )

Read the PID Output and Set the Motor Speeds.

**Examples:**

uartFlight.c.

Definition at line 69 of file set.c.

References baseSpeed, MAXDIFF, o_output, o_pids, and setMotorSpeeds().

```
69                     {
70     for (uint8_t i = 0; i < 2; i++) {
71         double diff = (o_output[i] * MAXDIFF) / o_pids[i].outMax;
72         uint8_t v[2] = { baseSpeed + diff, baseSpeed - diff };
73         setMotorSpeeds(i, v);
74     }
75 }
```

### 5.17.4 Variable Documentation

#### 5.17.4.1 uint8_t baseSpeed

Motor Base Speed.

**Examples:**

uartFlight.c.

Definition at line 53 of file set.c.

Referenced by setTask().

#### 5.17.4.2 uint8_t baseSpeed = 0

Motor Base Speed.

Definition at line 53 of file set.c.

Referenced by setTask().

## 5.18 SPI Driver

SPI Library for AVRs built-in SPI Hardware.

### Files

- file spi.h

  *SPI API Header.*
- file spi.c

  *SPI API Implementation.*

### Enumerations

- enum SPI_MODE { MODE_0 = 0, MODE_1 = 1, MODE_2 = 2, MODE_3 = 3 }

  *SPI Mode option.*
- enum SPI_SPEED {
  SPEED_2 = 4, SPEED_4 = 0, SPEED_8 = 5, SPEED_16 = 1,
  SPEED_32 = 6, SPEED_64 = 2, SPEED_128 = 3 }

  *SPI Speed options.*

### Functions

- void spiInit (SPI_MODE mode, SPI_SPEED speed)

  *Initialize the SPI Hardware Module.*
- uint8_t spiSendByte (uint8_t d)

  *Send and Receive one byte.*

### 5.18.1 Detailed Description

SPI Library for AVRs built-in SPI Hardware.

### 5.18.2 Enumeration Type Documentation

#### 5.18.2.1 enum SPI_MODE

SPI Mode option.

**Enumerator**

***MODE_0*** CPOL 0, CPHA 0.

***MODE_1*** CPOL 0, CPHA 1.

***MODE_2*** CPOL 1, CPHA 0.

***MODE_3*** CPOL 1, CPHA 1.

Definition at line 44 of file spi.h.

```
44             {
45     MODE_0 = 0,
46     MODE_1 = 1,
47     MODE_2 = 2,
48     MODE_3 = 3,
49 } SPI_MODE;
```

#### 5.18.2.2 enum SPI_SPEED

SPI Speed options.

**Enumerator**

***SPEED_2*** F_CPU / 2.

***SPEED_4*** F_CPU / 4.

***SPEED_8*** F_CPU / 8.

***SPEED_16*** F_CPU / 16.

***SPEED_32*** F_CPU / 32.

***SPEED_64*** F_CPU / 64.

***SPEED_128*** F_CPU / 128.

Definition at line 52 of file spi.h.

```
52              {
53     SPEED_2 = 4,
54     SPEED_4 = 0,
55     SPEED_8 = 5,
56     SPEED_16 = 1,
57     SPEED_32 = 6,
58     SPEED_64 = 2,
59     SPEED_128 = 3,
60 } SPI_SPEED;
```

### 5.18.3 Function Documentation

#### 5.18.3.1 void spiInit ( SPI_MODE *mode*, SPI_SPEED *speed* )

Initialize the SPI Hardware Module.

**Parameters**

| | |
|---|---|
| *mode* | SPI Mode to use |
| *speed* | SPI Speed to use |

Referenced by xyInit().

#### 5.18.3.2 uint8_t spiSendByte ( uint8_t *d* )

Send and Receive one byte.

Set the Chip Select Lines yourself!

**Parameters**

| | |
|---|---|
| *d* | Data to be sent |

**Returns**

Byte read from Bus

Definition at line 54 of file spi.c.

```
54                                {
55     SPDR = d;
56     while (!(SPSR & (1 << SPIF))); // Wait for transmission
57     return SPDR;
58 }
```

## 5.19 Task Handler

System for registering different tasks that will be called regularly, one after another.

### Files

- file tasks.h

  *Task API Header.*
- file tasks.c

  *Task API Implementation.*

### Data Structures

- struct TaskElement

  *Single-Linked Task List.*

### Typedefs

- typedef void(∗ Task )(void)

  *A Task has no arguments and returns nothing.*

### Functions

- uint8_t addTask (Task func)

  *Adds another task that will be called regularly.*
- uint8_t removeTask (Task func)

  *Removes an already registered Task.*
- void tasks (void)

  *Executes registered Tasks.*
- uint8_t tasksRegistered (void)

  *Get the number of registered Tasks.*

### Variables

- TaskElement ∗ taskList

  *List of registered Tasks.*
- TaskElement ∗ taskList = NULL

  *List of registered Tasks.*

### 5.19.1 Detailed Description

System for registering different tasks that will be called regularly, one after another.

### 5.19.2 Typedef Documentation

#### 5.19.2.1 typedef void(∗ Task)(void)

A Task has no arguments and returns nothing.

Definition at line 44 of file tasks.h.

### 5.19.3 Function Documentation

#### 5.19.3.1 uint8_t addTask ( Task *func* )

Adds another task that will be called regularly.

**Parameters**

| | |
|---|---|
| *func* | Task to be executed |

**Returns**

0 on success

**Examples:**

uartFlight.c.

Definition at line 57 of file tasks.c.

References BANK_GENERIC, MEMSWITCH, MEMSWITCHBACK, TaskElement::next, TaskElement::task, and taskList.

Referenced by xyInit().

```
57                              {
58     MEMSWITCH(BANK_GENERIC);
59     TaskElement *p = (TaskElement *)malloc(sizeof(
       TaskElement));
60     if (p == NULL) {
61         MEMSWITCHBACK(BANK_GENERIC);
62         return 1;
63     }
64     p->task = func;
65     p->next = taskList;
66     taskList = p;
67     MEMSWITCHBACK(BANK_GENERIC);
68     return 0;
69 }
```

#### 5.19.3.2 uint8_t removeTask ( Task *func* )

Removes an already registered Task.

**Parameters**

| | |
|---|---|
| *func* | Task to be removed |

**Returns**

0 on success

Definition at line 71 of file tasks.c.

References BANK_GENERIC, MEMSWITCH, MEMSWITCHBACK, TaskElement::next, TaskElement::task, and taskList.

```
71                                 {
72     MEMSWITCH(BANK_GENERIC);
73     TaskElement *p = taskList;
74     TaskElement *prev = NULL;
75     while (p != NULL) {
76         if (p->task == func) {
77             if (prev == NULL) {
78                 taskList = p->next;
79             } else {
```

```
80                 prev->next = p->next;
81             }
82             free(p);
83             MEMSWITCHBACK(BANK_GENERIC);
84             return 0;
85         }
86         prev = p;
87         p = p->next;
88     }
89     MEMSWITCHBACK(BANK_GENERIC);
90     return 1;
91 }
```

#### 5.19.3.3 void tasks ( void )

Executes registered Tasks.

Call this in your Main Loop!

**Examples:**

uartFlight.c.

Definition at line 93 of file tasks.c.

References BANK_GENERIC, MEMSWITCH, MEMSWITCHBACK, TaskElement::next, TaskElement::task, and taskList.

```
93                 {
94     MEMSWITCH(BANK_GENERIC);
95     static TaskElement *p = NULL;
96     if (p == NULL) {
97         p = taskList;
98     }
99     if (p != NULL) {
100         p->task();
101         p = p->next;
102     }
103     MEMSWITCHBACK(BANK_GENERIC);
104 }
```

#### 5.19.3.4 uint8_t tasksRegistered ( void )

Get the number of registered Tasks.

**Returns**

Count of registered Tasks

Definition at line 47 of file tasks.c.

References BANK_GENERIC, MEMSWITCH, MEMSWITCHBACK, and TaskElement::next.

```
47                                 {
48     uint8_t c = 0;
49     MEMSWITCH(BANK_GENERIC);
50     for (TaskElement *p = taskList; p != NULL; p = p->next) {
51         c++;
52     }
53     MEMSWITCHBACK(BANK_GENERIC);
54     return c;
55 }
```

### 5.19.4 Variable Documentation

#### 5.19.4.1 TaskElement* taskList = NULL

List of registered Tasks.

Definition at line 45 of file tasks.c.

Referenced by addTask(), removeTask(), and tasks().

#### 5.19.4.2 TaskElement* taskList

List of registered Tasks.

Definition at line 45 of file tasks.c.

Referenced by addTask(), removeTask(), and tasks().

## 5.20 Time Keeping

Measuring Time with Millisecond Resolution.

### Files

- file time.h
  *Time API Header.*
- file time.c
  *Time API Implementation.*

### Macros

- #define TCRA TCCR2A
  *Timer 2 Control Register A.*
- #define TCRB TCCR2B
  *Timer 2 Control Register B.*
- #define OCR OCR2A
  *Timer 2 Compare Register A.*
- #define TIMS TIMSK2
  *Timer 2 Interrupt Mask.*
- #define OCIE OCIE2A
  *Timer 2 Compare Match A Interrupt Enable.*

### Typedefs

- typedef uint64_t time_t
  *Timekeeping Data Type.*

### Functions

- void initSystemTimer (void)
  *Initialize the system timer.*
- time_t getSystemTime (void)
  *Get the System Uptime.*
- ISR (TIMER2_COMPA_vect)
  *Timer 2 Compare Match A Interrupt.*

### Variables

- volatile time_t systemTime = 0
  *Current System Uptime.*

### 5.20.1 Detailed Description

Measuring Time with Millisecond Resolution. Uses Timer 2

Prescaler 64

Count to 250

16000000 / 64 / 250 = 1000 –> 1 Interrupt per millisecond

### 5.20.2 Macro Definition Documentation

#### 5.20.2.1 #define OCIE OCIE2A

Timer 2 Compare Match A Interrupt Enable.

Definition at line 53 of file time.c.

#### 5.20.2.2 #define OCR OCR2A

Timer 2 Compare Register A.

Definition at line 51 of file time.c.

#### 5.20.2.3 #define TCRA TCCR2A

Timer 2 Control Register A.

Definition at line 49 of file time.c.

#### 5.20.2.4 #define TCRB TCCR2B

Timer 2 Control Register B.

Definition at line 50 of file time.c.

#### 5.20.2.5 #define TIMS TIMSK2

Timer 2 Interrupt Mask.

Definition at line 52 of file time.c.

### 5.20.3 Typedef Documentation

#### 5.20.3.1 typedef uint64_t time_t

Timekeeping Data Type.

Overflows after 500 million years... :)

Definition at line 53 of file time.h.

### 5.20.4 Function Documentation

#### 5.20.4.1 time_t getSystemTime ( void )

Get the System Uptime.

**Returns**

System Uptime in Milliseconds

**Examples:**

uartFlight.c.

Definition at line 68 of file time.c.

References systemTime.

Referenced by pidExecute().

```
68                              {
69     return systemTime;
70 }
```

#### 5.20.4.2 void initSystemTimer ( void )

Initialize the system timer.

Execution every millisecond. Uses Timer 2.

Definition at line 55 of file time.c.

Referenced by xyInit().

```
55                          {
56     // Timer initialization
57     TCRA |= (1 << WGM21); // CTC Mode
58     TCRB |= (1 << CS22); // Prescaler: 64
59     OCR = 250;
60     TIMS |= (1 << OCIE); // Enable compare match interrupt
61 }
```

#### 5.20.4.3 ISR ( TIMER2_COMPA_vect )

Timer 2 Compare Match A Interrupt.

Definition at line 64 of file time.c.

References systemTime.

```
64                          {
65     systemTime++;
66 }
```

### 5.20.5 Variable Documentation

#### 5.20.5.1 volatile time_t systemTime = 0

Current System Uptime.

Definition at line 47 of file time.c.

Referenced by getSystemTime(), and ISR().

## 5.21 I2C Driver

Using the AVR TWI/I2C Hardware.

### Files

- file twi.h

  *I2C API Header.*

### Macros

- #define TWI_READ 1

  *I2C Read Bit.*
- #define TWI_WRITE 0

  *I2C Write Bit.*

### Functions

- void twiInit (void)

  *Initialize the I2C Hardware.*
- void twiStop (void)

  *Stop the I2C Hardware.*
- unsigned char twiStart (unsigned char addr)

  *Start an I2C Transfer.*
- unsigned char twiRepStart (unsigned char addr)

  *Start a repeated I2C Transfer.*
- void twiStartWait (unsigned char addr)

  *Start an I2C Transfer and poll until ready.*
- unsigned char twiWrite (unsigned char data)

  *Write to the I2C Slave.*
- unsigned char twiReadAck (void)

  *Read from the I2C Slave and request more data.*
- unsigned char twiReadNak (void)

  *Read from the I2C Slave and deny more data.*

### 5.21.1 Detailed Description

Using the AVR TWI/I2C Hardware.

### 5.21.2 Macro Definition Documentation

#### 5.21.2.1 #define TWI_READ 1

I2C Read Bit.

Definition at line 43 of file twi.h.

Referenced by accRead(), gyroRead(), and magRead().

#### 5.21.2.2 #define TWI_WRITE 0

I2C Write Bit.

Definition at line 44 of file twi.h.

Referenced by accRead(), gyroRead(), magRead(), and motorTask().

### 5.21.3 Function Documentation

#### 5.21.3.1 void twiInit ( void )

Initialize the I2C Hardware.

Definition at line 26 of file twi.c.

Referenced by xyInit().

```
27 {
28   /* initialize TWI clock: 100 kHz clock, TWPS = 0 => prescaler = 1 */
29
30   TWSR = 0;                         /* no prescaler */
31   TWBR = ((F_CPU/SCL_CLOCK)-16)/2;  /* must be > 10 for stable operation */
32
33 }/* i2c_init */
```

#### 5.21.3.2 unsigned char twiReadAck ( void )

Read from the I2C Slave and request more data.

**Returns**

Data read

Definition at line 179 of file twi.c.

Referenced by accRead(), gyroRead(), and magRead().

```
180 {
181     TWCR = (1<<TWINT) | (1<<TWEN) | (1<<TWEA);
182     while(!(TWCR & (1<<TWINT)));
183
184     return TWDR;
185
186 }/* i2c_readAck */
```

#### 5.21.3.3 unsigned char twiReadNak ( void )

Read from the I2C Slave and deny more data.

**Returns**

Data read

Definition at line 194 of file twi.c.

Referenced by accRead(), gyroRead(), and magRead().

```
195 {
196     TWCR = (1<<TWINT) | (1<<TWEN);
197     while(!(TWCR & (1<<TWINT)));
198
199     return TWDR;
200
201 }/* i2c_readNak */
```

#### 5.21.3.4 unsigned char twiRepStart ( unsigned char *addr* )

Start a repeated I2C Transfer.

**Parameters**

| | |
|---|---|
| *addr* | Slave Address (with Read/Write bit) |

**Returns**

0 on success, 1 on error

Definition at line 127 of file twi.c.

References twiStart().

Referenced by accRead(), gyroRead(), and magRead().

```
128 {
129     return twiStart( address );
130
131 }/* i2c_rep_start */
```

#### 5.21.3.5 unsigned char twiStart ( unsigned char *addr* )

Start an I2C Transfer.

**Parameters**

| | |
|---|---|
| *addr* | Slave Address (with Read/Write bit) |

**Returns**

0 on success, 1 on error

Definition at line 40 of file twi.c.

Referenced by accRead(), gyroRead(), magRead(), motorTask(), and twiRepStart().

```
41 {
42     uint8_t   twst;
43
44     // send START condition
45     TWCR = (1<<TWINT) | (1<<TWSTA) | (1<<TWEN);
46
47     // wait until transmission completed
48     while(!(TWCR & (1<<TWINT)));
49
50     // check value of TWI Status Register. Mask prescaler bits.
51     twst = TW_STATUS & 0xF8;
52     if ( (twst != TW_START) && (twst != TW_REP_START)) return 1;
53
54     // send device address
55     TWDR = address;
56     TWCR = (1<<TWINT) | (1<<TWEN);
57
58     // wail until transmission completed and ACK/NACK has been received
59     while(!(TWCR & (1<<TWINT)));
60
61     // check value of TWI Status Register. Mask prescaler bits.
62     twst = TW_STATUS & 0xF8;
63     if ( (twst != TW_MT_SLA_ACK) && (twst != TW_MR_SLA_ACK) ) return 1;
64
65     return 0;
66
67 }/* i2c_start */
```

#### 5.21.3.6 void twiStartWait ( unsigned char *addr* )

Start an I2C Transfer and poll until ready.

**Parameters**

| | |
|---|---|
| *addr* | Slave Address (with Read/Write bit) |

Definition at line 76 of file twi.c.

```
77 {
78     uint8_t   twst;
79 
80 
81     while ( 1 )
82     {
83         // send START condition
84         TWCR = (1<<TWINT) | (1<<TWSTA) | (1<<TWEN);
85 
86         // wait until transmission completed
87         while(!(TWCR & (1<<TWINT)));
88 
89         // check value of TWI Status Register. Mask prescaler bits.
90         twst = TW_STATUS & 0xF8;
91         if ( (twst != TW_START) && (twst != TW_REP_START)) continue;
92 
93         // send device address
94         TWDR = address;
95         TWCR = (1<<TWINT) | (1<<TWEN);
96 
97         // wail until transmission completed
98         while(!(TWCR & (1<<TWINT)));
99 
100         // check value of TWI Status Register. Mask prescaler bits.
101         twst = TW_STATUS & 0xF8;
102         if ( (twst == TW_MT_SLA_NACK )||(twst ==TW_MR_DATA_NACK) )
103         {
104             /* device busy, send stop condition to terminate write operation */
105             TWCR = (1<<TWINT) | (1<<TWEN) | (1<<TWSTO);
106 
107             // wait until stop condition is executed and bus released
108             while(TWCR & (1<<TWSTO));
109 
110             continue;
111         }
112         //if( twst != TW_MT_SLA_ACK) return 1;
113         break;
114      }
115 
116 }/* i2c_start_wait */
```

#### 5.21.3.7 void twiStop ( void )

Stop the I2C Hardware.

Definition at line 137 of file twi.c.

Referenced by motorTask().

```
138 {
139     /* send stop condition */
140     TWCR = (1<<TWINT) | (1<<TWEN) | (1<<TWSTO);
141 
142     // wait until stop condition is executed and bus released
143     while(TWCR & (1<<TWSTO));
144 
145 }/* i2c_stop */
```

#### 5.21.3.8 unsigned char twiWrite ( unsigned char *data* )

Write to the I2C Slave.

**Parameters**

| | |
|---|---|
| *data* | Data to send |

**Returns**

0 on success, 1 on error

Definition at line 155 of file twi.c.

Referenced by accRead(), gyroRead(), magRead(), and motorTask().

```
156 {
157     uint8_t   twst;
158 
159     // send data to the previously addressed device
160     TWDR = data;
161     TWCR = (1<<TWINT) | (1<<TWEN);
162 
163     // wait until transmission completed
164     while(!(TWCR & (1<<TWINT)));
165 
166     // check value of TWI Status Register. Mask prescaler bits
167     twst = TW_STATUS & 0xF8;
168     if( twst != TW_MT_DATA_ACK) return 1;
169     return 0;
170 
171 }/* i2c_write */
```

## 5.22 UART Menu

Enables user interaction with an UART Menu.

### Files

- file uartMenu.h

  *UART Menu API Header.*
- file uartMenu.c

  *UART Menu API Implementation.*

### Data Structures

- struct MenuEntry

  *Data Structure for Single-Linked-List for UART Menu.*

### Functions

- uint8_t addMenuCommand (uint8_t cmd, PGM_P help, Task f)

  *Add a command to the UART Menu.*
- void uartMenuPrintHelp (void)

  *Print all registered commands.*
- void uartMenuRegisterHandler (void(∗handler)(char))

  *Register a Handler for unhandled menu commands.*
- void uartMenuTask (void)

  *Task to work the UART Menu.*
- MenuEntry ∗ findEntry (uint8_t cmd)

  *Search the uartMenu Linked List.*
- MenuEntry ∗ reverseList (MenuEntry ∗root)

  *Reverse the UART Menu List.*

### Variables

- MenuEntry ∗ uartMenu = NULL

  *Single-Linked-List for commands.*
- void(∗ unHandler )(char) = NULL

  *Handler for unhandled commands.*

### 5.22.1 Detailed Description

Enables user interaction with an UART Menu.

### 5.22.2 Function Documentation

#### 5.22.2.1 uint8_t addMenuCommand ( uint8_t *cmd*, PGM_P *help*, Task *f* )

Add a command to the UART Menu.

**Parameters**

| | |
|---|---|
| *cmd* | Byte that triggers command |
| *help* | Help Text String in Flash |
| *f* | Task to be executed |

**Returns**

0 on success, 1 if already registered or not enough memory.

**Examples:**

uartFlight.c.

Definition at line 69 of file uartMenu.c.

References BANK_GENERIC, MenuEntry::cmd, MenuEntry::f, findEntry(), MenuEntry::helpText, MenuEntry::next, uartMenu, xmemGetBank(), and xmemSetBank().

Referenced by xyInit().

```
69                                                      {
70     uint8_t lastBank = xmemGetBank();
71     xmemSetBank(BANK_GENERIC);
72     if (findEntry(cmd) != NULL) {
73         return 1;
74     } else {
75         MenuEntry *p = (MenuEntry *)malloc(sizeof(MenuEntry));
76         if (p == NULL) {
77             return 1;
78         }
79         p->cmd = cmd;
80         p->helpText = help;
81         p->f = f;
82         p->next = uartMenu;
83         uartMenu = p;
84         return 0;
85     }
86     xmemSetBank(lastBank);
87 }
```

#### 5.22.2.2 MenuEntry* findEntry ( uint8_t *cmd* )

Search the uartMenu Linked List.

**Parameters**

| | |
|---|---|
| *cmd* | Command to search for |

**Returns**

MenuEntry for command cmd, or NULL

Definition at line 58 of file uartMenu.c.

References MenuEntry::cmd, MenuEntry::next, and uartMenu.

Referenced by addMenuCommand().

```
58                                      {
59     MenuEntry *p = uartMenu;
60     while (p != NULL) {
61         if (p->cmd == cmd) {
62             return p;
63         }
64         p = p->next;
65     }
66     return NULL;
67 }
```

#### 5.22.2.3 MenuEntry* reverseList ( MenuEntry * *root* )

Reverse the UART Menu List.

**Parameters**

| | |
|---|---|
| *root* | Root of the Single-Linked-List. |

**Returns**

New root of reversed list.

Definition at line 93 of file uartMenu.c.

References MenuEntry::next.

Referenced by uartMenuPrintHelp().

```
93                                          {
94     MenuEntry *new = NULL;
95     while (root != NULL) {
96         MenuEntry *next = root->next;
97         root->next = new;
98         new = root;
99         root = next;
100    }
101    return new;
102 }
```

#### 5.22.2.4 void uartMenuPrintHelp ( void )

Print all registered commands.

Definition at line 104 of file uartMenu.c.

References BANK_GENERIC, MenuEntry::cmd, MenuEntry::helpText, MenuEntry::next, reverseList(), uartMenu, xmemGetBank(), and xmemSetBank().

Referenced by xyInit().

```
104                             {
105     static uint8_t reversed = 0;
106     uint8_t lastBank = xmemGetBank();
107     xmemSetBank(BANK_GENERIC);
108     char *buffer = (char *)malloc(35);
109     if (buffer == NULL) {
110         printf("!");
111         return;
112     }
113     if (!reversed) {
114         reversed = 1;
115         uartMenu = reverseList(uartMenu);
116     }
117     MenuEntry *p = uartMenu;
118     while (p != NULL) {
119         strcpy_P(buffer, p->helpText);
120         printf("%c: %s\n", p->cmd, buffer);
121         p = p->next;
122     }
123     free(buffer);
124     xmemSetBank(lastBank);
125 }
```

#### 5.22.2.5 void uartMenuRegisterHandler ( void(*)(char) *handler* )

Register a Handler for unhandled menu commands.

**Parameters**

| | |
|---|---|
| *handler* | Will be called if an unknown command is received. |

Definition at line 127 of file uartMenu.c.

References unHandler.

```
127                                                 {
128     unHandler = handler;
129 }
```

#### 5.22.2.6 void uartMenuTask ( void )

Task to work the UART Menu.

Definition at line 131 of file uartMenu.c.

References BANK_GENERIC, MenuEntry::cmd, MenuEntry::f, MenuEntry::next, serialGet(), serialHasChar(), uartMenu, unHandler, xmemGetBank(), and xmemSetBank().

Referenced by xyInit().

```
131                         {
132     if (serialHasChar()) {
133         uint8_t lastBank = xmemGetBank();
134         xmemSetBank(BANK_GENERIC);
135         uint8_t c = serialGet();
136         MenuEntry *p = uartMenu;
137         while (p != NULL) {
138             if (p->cmd == c) {
139                 p->f();
140                 xmemSetBank(lastBank);
141                 return;
142             }
143             p = p->next;
144         }
145         if (unHandler != NULL)
146             unHandler(c);
147         xmemSetBank(lastBank);
148     }
149 }
```

### 5.22.3 Variable Documentation

#### 5.22.3.1 MenuEntry* uartMenu = NULL

Single-Linked-List for commands.

Definition at line 51 of file uartMenu.c.

Referenced by addMenuCommand(), findEntry(), uartMenuPrintHelp(), and uartMenuTask().

#### 5.22.3.2 void(* unHandler)(char) = NULL

Handler for unhandled commands.

Definition at line 52 of file uartMenu.c.

Referenced by uartMenuRegisterHandler(), and uartMenuTask().

## 5.23 External Memory Interface

Allows access to external RAM with bank-switching.

### Files

- file xmem.h

  *XMEM API Header.*
- file xmem.c

  *XMEM API Implementation.*

### Data Structures

- struct MallocState

  *All Malloc related State.*

### Macros

- #define MEMSWITCH(x) uint8_t oldMemBank=xmemGetBank();if(oldMemBank!=x)xmemSetBank(x);

  *Switch the bank, if needed.*
- #define MEMSWITCHBACK(x) if(oldMemBank!=x)xmemSetBank(oldMemBank);

  *Switch back to the last bank, if needed.*
- #define MEMBANKS 8

  *Available Memory Banks.*
- #define BANK_GENERIC 0

  *Generic Memory Bank.*

### Functions

- void xmemInit (void)

  *Initialize the External Memory Interface.*
- void xmemSetBank (uint8_t bank)

  *Switch the active memory bank.*
- uint8_t xmemGetBank (void)

  *Get the current memory bank.*
- void saveState (uint8_t bank)

  *Save the current malloc state.*
- void restoreState (uint8_t bank)

  *Restore the malloc state.*

### Variables

- MallocState states [MEMBANKS]

  *MallocState for all Memory Banks.*
- uint8_t currentBank

  *Current active Memory Bank.*
- MallocState states [MEMBANKS]

  *MallocState for all Memory Banks.*
- uint8_t currentBank = 0

*Current active Memory Bank.*

- void * __brkval

  *Internal Malloc Heap-End Pointer.*

- void * __flp

  *Internal Malloc Free List Pointer (State)*

### 5.23.1 Detailed Description

Allows access to external RAM with bank-switching.

### 5.23.2 Macro Definition Documentation

#### 5.23.2.1 #define BANK_GENERIC 0

Generic Memory Bank.

Definition at line 55 of file xmem.h.

Referenced by addMenuCommand(), addTask(), removeTask(), tasks(), tasksRegistered(), uartMenuPrintHelp(), and uartMenuTask().

#### 5.23.2.2 #define MEMBANKS 8

Available Memory Banks.

Definition at line 54 of file xmem.h.

Referenced by xmemInit(), and xmemSetBank().

#### 5.23.2.3 #define MEMSWITCH( *x* ) uint8_t oldMemBank=**xmemGetBank**();if(oldMemBank!=x)**xmemSetBank**(x);

Switch the bank, if needed.

Stores the old bank in a variable oldMemBank.

**Parameters**

| | |
|---|---|
| *x* | New Bank |

Definition at line 47 of file xmem.h.

Referenced by addTask(), removeTask(), tasks(), and tasksRegistered().

#### 5.23.2.4 #define MEMSWITCHBACK( *x* ) if(oldMemBank!=x)**xmemSetBank**(oldMemBank);

Switch back to the last bank, if needed.

**Parameters**

| | |
|---|---|
| *x* | New (current) Bank |

Definition at line 52 of file xmem.h.

Referenced by addTask(), removeTask(), tasks(), and tasksRegistered().

### 5.23.3 Function Documentation

#### 5.23.3.1 void restoreState ( uint8_t *bank* )

Restore the malloc state.

**Parameters**

| | |
|---|---|
| *bank* | Location of state to load. |

Definition at line 65 of file xmem.c.

References __brkval, __flp, MallocState::end, MallocState::fl, MallocState::start, and MallocState::val.

Referenced by xmemSetBank().

```
65                                  {
66     __malloc_heap_start = states[bank].start;
67     __malloc_heap_end = states[bank].end;
68     __brkval = states[bank].val;
69     __flp = states[bank].fl;
70 }
```

#### 5.23.3.2 void saveState ( uint8_t *bank* )

Save the current malloc state.

**Parameters**

| | |
|---|---|
| *bank* | Current Bank Number |

Definition at line 55 of file xmem.c.

References __brkval, __flp, MallocState::end, MallocState::fl, MallocState::start, and MallocState::val.

Referenced by xmemInit(), and xmemSetBank().

```
55                                  {
56     states[bank].start = __malloc_heap_start;
57     states[bank].end = __malloc_heap_end;
58     states[bank].val = __brkval;
59     states[bank].fl = __flp;
60 }
```

#### 5.23.3.3 uint8_t xmemGetBank ( void )

Get the current memory bank.

**Returns**

Current Memory Bank.

Definition at line 105 of file xmem.c.

References currentBank.

Referenced by addMenuCommand(), uartMenuPrintHelp(), and uartMenuTask().

```
105                                 {
106     return currentBank;
107 }
```

#### 5.23.3.4 void xmemInit ( void )

Initialize the External Memory Interface.

Definition at line 72 of file xmem.c.

References BANK0DDR, BANK0PIN, BANK0PORT, BANK1DDR, BANK1PIN, BANK1PORT, BANK2DDR, BANK2PIN, BANK2PORT, MEMBANKS, and saveState().

Referenced by xyInit().

```
72                      {
73     BANK0DDR |= (1 << BANK0PIN);
74     BANK1DDR |= (1 << BANK1PIN);
75     BANK2DDR |= (1 << BANK2PIN);
76     BANK0PORT &= ~(1 << BANK0PIN);
77     BANK1PORT &= ~(1 << BANK1PIN);
78     BANK2PORT &= ~(1 << BANK2PIN);
79
80     XMCRB = 0; // Use full address space
81     XMCRA = (1 << SRW11) | (1 << SRW10); // 3 Wait cycles
82     XMCRA |= (1 << SRE); // Enable XMEM
83
84     for (uint8_t i = 0; i < MEMBANKS; i++) {
85         saveState(i);
86     }
87 }
```

#### 5.23.3.5 void xmemSetBank ( uint8_t *bank* )

Switch the active memory bank.

**Parameters**

| | |
|---|---|
| *bank* | New Memory Bank |

Definition at line 89 of file xmem.c.

References BANK0PIN, BANK0PORT, BANK1PIN, BANK1PORT, BANK2PIN, BANK2PORT, currentBank, MEMBANKS, restoreState(), and saveState().

Referenced by addMenuCommand(), uartMenuPrintHelp(), and uartMenuTask().

```
89                              {
90     if (bank < MEMBANKS) {
91         saveState(currentBank);
92
93         BANK0PORT &= ~(1 << BANK0PIN);
94         BANK1PORT &= ~(1 << BANK1PIN);
95         BANK2PORT &= ~(1 << BANK2PIN);
96         BANK0PORT |= ((bank & 0x01) << BANK0PIN);
97         BANK1PORT |= (((bank & 0x02) >> 1) << BANK1PIN);
98         BANK2PORT |= (((bank & 0x04) >> 2) << BANK2PIN);
99
100         currentBank = bank;
101         restoreState(bank);
102     }
103 }
```

### 5.23.4 Variable Documentation

#### 5.23.4.1 void* __brkval

Internal Malloc Heap-End Pointer.

Referenced by restoreState(), and saveState().

#### 5.23.4.2 void* __flp

Internal Malloc Free List Pointer (State)

Referenced by restoreState(), and saveState().

#### 5.23.4.3 uint8_t currentBank = 0

Current active Memory Bank.

Definition at line 47 of file xmem.c.

Referenced by xmemGetBank(), and xmemSetBank().

#### 5.23.4.4 uint8_t currentBank

Current active Memory Bank.

Definition at line 47 of file xmem.c.

Referenced by xmemGetBank(), and xmemSetBank().

#### 5.23.4.5 MallocState states[MEMBANKS]

MallocState for all Memory Banks.

Definition at line 46 of file xmem.c.

#### 5.23.4.6 MallocState states[MEMBANKS]

MallocState for all Memory Banks.

Definition at line 46 of file xmem.c.

## 5.24 xyControl Hardware

Controls xyControl On-Board Hardware like LEDs.

### Files

- file xycontrol.h
  *xyControl API Header.*
- file xycontrol.c
  *xyControl API Implementation.*

### Data Structures

- struct Vector3f
  *The global 3-Dimensional Floating Point Vector.*

### Enumerations

- enum LED {
  LED_RED0 = 0, LED_RED1 = 1, LED_GREEN0 = 2, LED_GREEN1 = 3,
  LED_ALL = 4, LED_BITMAP = 5, LED_RED = 6, LED_GREEN = 7 }
  *Methods of addressing the LEDs.*
- enum LEDState { LED_OFF = 0, LED_ON = 1, LED_TOGGLE = 2 }
  *Possible states of the LEDs.*

### Functions

- void xyInit (void)
  *Initialize the xyControl Hardware.*
- void xyLed (LED l, LEDState v)
  *Set the LEDs.*
- double getVoltage (void)
  *Calculate and return the Battery Voltage.*
- void resetSelf (void)
  *Use the Watchdog to reset yourself after 15ms.*
- int uartoutput (char c, FILE ∗f)
  *Method used to write to stdout and stderr.*
- int uartinput (FILE ∗f)
  *Method used to read from stdin.*
- void xyLedInternal (uint8_t v, volatile uint8_t ∗port, uint8_t pin)
  *Internal LED Manipulation function.*

### Variables

- char PROGMEM helpText [] = "Print this Help"
  *UART Menu Help Text.*
- char PROGMEM resetText [] = "Reset MCU"
  *UART Menu Reset Text.*
- FILE inFile
  *FILE for stdin.*
- FILE outFile
  *FILE for stdout and stderr.*

### 5.24.1 Detailed Description

Controls xyControl On-Board Hardware like LEDs.

### 5.24.2 Enumeration Type Documentation

#### 5.24.2.1 enum LED

Methods of addressing the LEDs.

**Enumerator**

***LED_RED0*** First red LED.

***LED_RED1*** Second red LED.

***LED_GREEN0*** First green LED.

***LED_GREEN1*** Second green LED.

***LED_ALL*** All LEDs.

***LED_BITMAP*** LEDs as Bitmap (R0, R1, G0, G1)

***LED_RED*** Both red LEDs.

***LED_GREEN*** Both green LEDs.

Definition at line 44 of file xycontrol.h.

```
44              {
45     LED_RED0 = 0,
46     LED_RED1 = 1,
47     LED_GREEN0 = 2,
48     LED_GREEN1 = 3,
49     LED_ALL = 4,
50     LED_BITMAP = 5,
51     LED_RED = 6,
52     LED_GREEN = 7
53 } LED;
```

#### 5.24.2.2 enum LEDState

Possible states of the LEDs.

**Enumerator**

***LED_OFF*** LED Off.

***LED_ON*** LED On.

***LED_TOGGLE*** Toggle the LED.

Definition at line 56 of file xycontrol.h.

```
56              {
57     LED_OFF = 0,
58     LED_ON = 1,
59     LED_TOGGLE = 2
60 } LEDState;
```

### 5.24.3 Function Documentation

#### 5.24.3.1 double getVoltage ( void )

Calculate and return the Battery Voltage.

**Returns**

Current Battery Voltage

**Examples:**

uartFlight.c.

Definition at line 163 of file xycontrol.c.

References adcGet(), adcReady(), adcStart(), BATT_CHANNEL, and BATT_MAX.

```
163                         {
164     adcStart(BATT_CHANNEL);
165     while(!adcReady());
166     uint16_t v = adcGet(0) * BATT_MAX;
167     return ((double)v / 1024.0);
168 }
```

#### 5.24.3.2 void resetSelf ( void )

Use the Watchdog to reset yourself after 15ms.

Definition at line 170 of file xycontrol.c.

Referenced by xyInit().

```
170                         {
171     wdt_enable(WDTO_15MS);
172     for(;;);
173 }
```

#### 5.24.3.3 int uartinput ( FILE * f )

Method used to read from stdin.

Definition at line 79 of file xycontrol.c.

References serialGet(), and serialHasChar().

Referenced by xyInit().

```
79                          {
80     while (!serialHasChar());
81     return serialGet();
82 }
```

#### 5.24.3.4 int uartoutput ( char c, FILE * f )

Method used to write to stdout and stderr.

Definition at line 66 of file xycontrol.c.

References serialWrite().

Referenced by xyInit().

```
66                                  {
67     // Inject CR here, instead of in the serial library,
68     // so we can still do binary transfers with serialWrite()...
69     if (c == '\n') {
70         serialWrite('\r');
71     }
72     if (c != '\r') {
73         serialWrite(c);
74     }
75     return 0;
76 }
```

#### 5.24.3.5 void xyInit ( void )

Initialize the xyControl Hardware.

Initializes LEDs, Timer, UART, I2C, SPI, ADC, the UART Menu and prepares stdin and stdout.

**Examples:**

uartFlight.c.

Definition at line 84 of file xycontrol.c.

References adcInit(), addMenuCommand(), addTask(), AVCC, BAUD, helpText, inFile, initSystemTimer(), LED0DDR, LED0PIN, LED1DDR, LED1PIN, LED2DDR, LED2PIN, LED3DDR, LED3PIN, MODE_0, outFile, resetSelf(), resetText, serialInit(), SPEED_2, spiInit(), twiInit(), uartinput(), uartMenuPrintHelp(), uartMenuTask(), uartoutput(), xmemInit(), and xyLed().

```
84                  {
85      xmemInit(); // Most important!
86
87      // LEDs
88      LED0DDR |= (1 << LED0PIN);
89      LED1DDR |= (1 << LED1PIN);
90      LED2DDR |= (1 << LED2PIN);
91      LED3DDR |= (1 << LED3PIN);
92      xyLed(4, 1);
93
94      initSystemTimer();
95      serialInit(BAUD(38400, F_CPU));
96      twiInit();
97      spiInit(MODE_0, SPEED_2);
98      adcInit(AVCC);
99
100      addMenuCommand('q', resetText, &resetSelf);
101      addMenuCommand('h', helpText, &uartMenuPrintHelp);
102      addTask(&uartMenuTask);
103
104      // fdevopen() is using malloc, so printf in a different
105      // memory bank will not work!
106      //   fdevopen(&uartoutput, NULL); // stdout & stderr
107      //   fdevopen(NULL, &uartinput); // stdin
108      // Instead we have the FILE structs as static variables
109      // and assign them to stdin, stdout and stderr
110
111      fdev_setup_stream(&outFile, &uartoutput, NULL, _FDEV_SETUP_WRITE);
112      fdev_setup_stream(&inFile, NULL, &uartinput, _FDEV_SETUP_READ);
113      stdin = &inFile;
114      stdout = &outFile;
115      stderr = &outFile;
116
117      sei();
118  }
```

#### 5.24.3.6 void xyLed ( LED *l*, LEDState *v* )

Set the LEDs.

**Parameters**

| | |
|---|---|
| *l* | LEDs to set |
| *v* | New LED State |

**Examples:**

uartFlight.c.

Referenced by xyInit().

#### 5.24.3.7 void xyLedInternal ( uint8_t *v*, volatile uint8_t * *port*, uint8_t *pin* )

Internal LED Manipulation function.

**Parameters**

| | |
|---|---|
| *v* | New LED State (Off, On, Toggle) |
| *port* | The Corresponding Output Port |
| *pin* | The LED Pin |

Definition at line 125 of file xycontrol.c.

```
125                                                                 {
126     if (v == 0) {
127         *port &= ~(1 << pin);
128     } else if (v == 1) {
129         *port |= (1 << pin);
130     } else {
131         *port ^= (1 << pin);
132     }
133 }
```

### 5.24.4 Variable Documentation

#### 5.24.4.1 char PROGMEM helpText[ ] = "Print this Help"

UART Menu Help Text.

Definition at line 59 of file xycontrol.c.

Referenced by xyInit().

#### 5.24.4.2 FILE inFile

FILE for stdin.

Definition at line 62 of file xycontrol.c.

Referenced by xyInit().

#### 5.24.4.3 FILE outFile

FILE for stdout and stderr.

Definition at line 63 of file xycontrol.c.

Referenced by xyInit().

#### 5.24.4.4 char PROGMEM resetText[ ] = "Reset MCU"

UART Menu Reset Text.

Definition at line 60 of file xycontrol.c.

Referenced by xyInit().

# Chapter 6

# Data Structure Documentation

## 6.1 Angles Struct Reference

Can store orientation in Euler Space.

`#include <orientation.h>`

### Data Fields

- double pitch

  *Pitch Angle in Degrees.*
- double roll

  *Roll Angle in Degrees.*
- double yaw

  *Yaw Angle in Degrees.*

### 6.1.1 Detailed Description

Can store orientation in Euler Space.

Definition at line 48 of file orientation.h.

### 6.1.2 Field Documentation

#### 6.1.2.1 double pitch

Pitch Angle in Degrees.

**Examples:**

uartFlight.c.

Definition at line 49 of file orientation.h.

Referenced by orientationTask(), pidTask(), and zeroOrientation().

#### 6.1.2.2 double roll

Roll Angle in Degrees.

**Examples:**

uartFlight.c.

Definition at line 50 of file orientation.h.

Referenced by orientationTask(), pidTask(), and zeroOrientation().

#### 6.1.2.3 double yaw

Yaw Angle in Degrees.

**Examples:**

uartFlight.c.

Definition at line 51 of file orientation.h.

Referenced by zeroOrientation().

The documentation for this struct was generated from the following file:

- include/orientation.h

## 6.2 Kalman Struct Reference

Kalman-Filter State data.

`#include <kalman.h>`

### Data Fields

- double x3

  *X Vector.*
- double p33

  *P Matrix.*

### 6.2.1 Detailed Description

Kalman-Filter State data.

Definition at line 47 of file kalman.h.

### 6.2.2 Field Documentation

#### 6.2.2.1 double p33

P Matrix.

Definition at line 49 of file kalman.h.

Referenced by kalmanInit(), and kalmanInnovate().

#### 6.2.2.2 double x3

X Vector.

Definition at line 48 of file kalman.h.

Referenced by kalmanInit(), and kalmanInnovate().

The documentation for this struct was generated from the following file:

- include/kalman.h

## 6.3 MallocState Struct Reference

All Malloc related State.

`#include <xmem.h>`

### Data Fields

- char * start

  *Start of Heap.*
- char * end

  *End of Heap.*
- void * val

  *Highest Heap Point.*
- void * fl

  *Free List.*

### 6.3.1 Detailed Description

All Malloc related State.

The Heap is bank-switched, so this state has to be switched with the banks to allow different memory allocations on different banks.

Definition at line 62 of file xmem.h.

### 6.3.2 Field Documentation

#### 6.3.2.1 char* end

End of Heap.

Definition at line 64 of file xmem.h.

Referenced by restoreState(), and saveState().

#### 6.3.2.2 void* fl

Free List.

Definition at line 66 of file xmem.h.

Referenced by restoreState(), and saveState().

#### 6.3.2.3 char* start

Start of Heap.

Definition at line 63 of file xmem.h.

Referenced by restoreState(), and saveState().

#### 6.3.2.4 void* val

Highest Heap Point.

Definition at line 65 of file xmem.h.

Referenced by restoreState(), and saveState().

The documentation for this struct was generated from the following file:

- include/xmem.h

## 6.4 MenuEntry Struct Reference

Data Structure for Single-Linked-List for UART Menu.

`#include <uartMenu.h>`

### Data Fields

- uint8_t cmd

  *Byte that triggers the action.*
- PGM_P helpText

  *Text (in Flash) printed with help command.*
- Task f

  *Action that get's executed.*
- MenuEntry * next

  *Next MenuEntry in the linked list.*

### 6.4.1 Detailed Description

Data Structure for Single-Linked-List for UART Menu.

Stores Helptext, command and action.

Definition at line 49 of file uartMenu.h.

### 6.4.2 Field Documentation

#### 6.4.2.1 uint8_t cmd

Byte that triggers the action.

Definition at line 50 of file uartMenu.h.

Referenced by addMenuCommand(), findEntry(), uartMenuPrintHelp(), and uartMenuTask().

#### 6.4.2.2 Task f

Action that get's executed.

Definition at line 52 of file uartMenu.h.

Referenced by addMenuCommand(), and uartMenuTask().

#### 6.4.2.3 PGM_P helpText

Text (in Flash) printed with help command.

Definition at line 51 of file uartMenu.h.

Referenced by addMenuCommand(), and uartMenuPrintHelp().

#### 6.4.2.4 MenuEntry* next

Next MenuEntry in the linked list.

Definition at line 53 of file uartMenu.h.

Referenced by addMenuCommand(), findEntry(), reverseList(), uartMenuPrintHelp(), and uartMenuTask().

The documentation for this struct was generated from the following file:

- include/uartMenu.h

## 6.5 PIDState Struct Reference

Data Structure for a single PID Controller.

`#include <pid.h>`

### Data Fields

- double kp
  *Proportional factor.*
- double ki
  *Integral factor.*
- double kd
  *Derivative factor.*
- double outMin
  *Minimum Output.*
- double outMax
  *Maximum Output.*
- double intMin
  *Minimum Integral sum.*
- double intMax
  *Maximum Integral sum.*
- double lastError
  *Derivative State.*
- double sumError
  *Integral state.*
- time_t last
  *Last execution time.*

### 6.5.1 Detailed Description

Data Structure for a single PID Controller.

Stores all needed constants and state variables.

Definition at line 47 of file pid.h.

### 6.5.2 Field Documentation

#### 6.5.2.1 double intMax

Maximum Integral sum.

Default is PID_INTMAX.

Definition at line 54 of file pid.h.

Referenced by pidExecute(), and pidSet().

#### 6.5.2.2 double intMin

Minimum Integral sum.

Default is PID_INTMIN.

Definition at line 53 of file pid.h.

Referenced by pidExecute(), and pidSet().

#### 6.5.2.3 double kd

Derivative factor.

Default is PID_D.

Definition at line 50 of file pid.h.

Referenced by pidExecute(), and pidSet().

#### 6.5.2.4 double ki

Integral factor.

Default is PID_I.

Definition at line 49 of file pid.h.

Referenced by pidExecute(), and pidSet().

#### 6.5.2.5 double kp

Proportional factor.

Default is PID_P.

Definition at line 48 of file pid.h.

Referenced by pidExecute(), and pidSet().

#### 6.5.2.6 time_t last

Last execution time.

For dT calculation.

Definition at line 57 of file pid.h.

Referenced by pidExecute(), and pidSet().

#### 6.5.2.7 double lastError

Derivative State.

Definition at line 55 of file pid.h.

Referenced by pidExecute(), and pidSet().

#### 6.5.2.8 double outMax

Maximum Output.

Default is PID_OUTMAX.

Definition at line 52 of file pid.h.

Referenced by pidExecute(), and pidSet().

#### 6.5.2.9 double outMin

Minimum Output.

Default is PID_OUTMIN.

Definition at line 51 of file pid.h.

Referenced by pidExecute(), and pidSet().

#### 6.5.2.10 double sumError

Integral state.

Kept in intMin, intMax Range.

Definition at line 56 of file pid.h.

Referenced by pidExecute(), and pidSet().

The documentation for this struct was generated from the following file:

- include/pid.h

## 6.6 TaskElement Struct Reference

Single-Linked Task List.

`#include <tasks.h>`

### Data Fields

- Task task

*Task to be executed.*

- TaskElement * next

  *Next list element.*

### 6.6.1 Detailed Description

Single-Linked Task List.

Definition at line 48 of file tasks.h.

### 6.6.2 Field Documentation

#### 6.6.2.1 TaskElement* next

Next list element.

Definition at line 50 of file tasks.h.

Referenced by addTask(), removeTask(), tasks(), and tasksRegistered().

#### 6.6.2.2 Task task

Task to be executed.

Definition at line 49 of file tasks.h.

Referenced by addTask(), removeTask(), and tasks().

The documentation for this struct was generated from the following file:

- include/tasks.h

## 6.7 Vector3f Struct Reference

The global 3-Dimensional Floating Point Vector.

`#include <xycontrol.h>`

### Data Fields

- double x

  *X Part.*

- double y

  *Y Part.*

- double z

  *Z Part.*

### 6.7.1 Detailed Description

The global 3-Dimensional Floating Point Vector.

Definition at line 63 of file xycontrol.h.

### 6.7.2 Field Documentation

#### 6.7.2.1 double x

X Part.

Definition at line 64 of file xycontrol.h.

Referenced by accRead(), gyroRead(), magRead(), and orientationTask().

#### 6.7.2.2 double y

Y Part.

Definition at line 65 of file xycontrol.h.

Referenced by accRead(), gyroRead(), magRead(), and orientationTask().

#### 6.7.2.3 double z

Z Part.

Definition at line 66 of file xycontrol.h.

Referenced by accRead(), gyroRead(), magRead(), and orientationTask().

The documentation for this struct was generated from the following file:

- include/xycontrol.h

# Chapter 7

# File Documentation

## 7.1 include/acc.h File Reference

LSM303DLHC Accelerometer API Header.

```
#include <error.h>
#include <xycontrol.h>
```

### Enumerations

- enum AccRange { r2G, r4G, r8G, r16G }

  *Accelerometer Range options.*

### Functions

- Error accInit (AccRange r)

  *Initialize the Accelerometer.*
- Error accRead (Vector3f *v)

  *Read from the Accelerometer.*

### 7.1.1 Detailed Description

LSM303DLHC Accelerometer API Header.

Definition in file acc.h.

## 7.2 include/adc.h File Reference

Analog-to-Digital Converter API Header.

### Enumerations

- enum ADCRef { AREF, AVCC, AINT1, AINT2 }

  *ADC Reference Voltage options.*

### Functions

- void adcInit (ADCRef ref)

  *Initialize the ADC Hardware.*
- void adcStart (uint8_t channel)

  *Start a conversion on a given channel.*
- uint8_t adcReady (void)

  *Check if a result is ready.*
- uint16_t adcGet (uint8_t next)

  *Get the conversion results.*
- void adcClose (void)

  *Disable the ADC to save energy.*

#### 7.2.1 Detailed Description

Analog-to-Digital Converter API Header.

Definition in file adc.h.

## 7.3 include/config.h File Reference

Various default settings.

### Macros

- #define SOFTWARELOWPASS 50

  *Software Low-Pass on Gyro and ACC.*
- #define ACCFILTERFACTOR SOFTWARELOWPASS

  *Accelerometer Low Pass Factor.*
- #define GYROFILTERFACTOR SOFTWARELOWPASS

  *Gyroscope Low Pass Factor.*
- #define PID_OUTMAX 256

  *Maximum PID Output.*
- #define PID_OUTMIN -256

  *Minimum PID Output.*
- #define PID_INTMAX PID_OUTMAX

  *Maximum PID Integral Sum.*
- #define PID_INTMIN PID_OUTMIN

  *Minimal PID Integral Sum.*
- #define PID_FACTOR 4 / 5

  *Influence of PID in relation to Base Speed.*
- #define DT 0.01f

  *Time Constant.*
- #define Q1 5.0f

  *Q Matrix Diagonal Element 1.*
- #define Q2 100.0f

  *Q Matrix Diagonal Element 2.*
- #define Q3 0.01f

  *Q Matrix Diagonal Element 3.*
- #define R1 1000.0f

*R Matrix Diagonal Element 1.*

- #define R2 1000.0f

  *R Matrix Diagonal Element 2.*
- #define SET_ROLLPLUS 1

  *Second Motor at the Right.*
- #define SET_ROLLMINUS 3

  *Fourth Motor at the Left.*
- #define SET_PITCHPLUS 0

  *First Motor at the Top.*
- #define SET_PITCHMINUS 2

  *Third Motor at the Bottom.*
- #define PID_P 5.0

  *Default PID P Constant.*
- #define PID_I 0.03

  *Default PID I Constant.*
- #define PID_D -13.0

  *Default PID D Constant.*
- #define MOTORCOUNT 4

  *Amount of motors.*
- #define BATT_MAX 15

  *Battery Voltage Reference (ADC 5V)*
- #define BATT_CHANNEL 0

  *ADC Channel for Battery.*
- #define ACC_ADDRESS 0x32

  *Accelerometer Address (0011001r)*
- #define GYRO_ADDRESS 0xD6

  *Gyroscope Address (110101xr, x = 1)*
- #define MAG_ADDRESS 0x3C

  *Magnetometer Address.*
- #define MOTOR_BASEADDRESS 0x52

  *Address of first motor controller.*
- #define LED0PORT PORTL

  *First LED Port.*
- #define LED0DDR DDRL

  *First LED Data Direction Register.*
- #define LED0PIN PL6

  *First LED Pin.*
- #define LED1PORT PORTL

  *Second LED Port.*
- #define LED1DDR DDRL

  *Second LED Data Direction Register.*
- #define LED1PIN PL7

  *Second LED Pin.*
- #define LED2PORT PORTG

  *Third LED Port.*
- #define LED2DDR DDRG

  *Third LED Data Direction Register.*
- #define LED2PIN PG5

  *Third LED Pin.*
- #define LED3PORT PORTE

  *Fourth LED Port.*

- #define LED3DDR DDRE

  *Fourth LED Data Direction Register.*
- #define LED3PIN PE2

  *Fourth LED Pin.*
- #define BANK0PORT PORTG

  *First Bank Selection Port.*
- #define BANK0DDR DDRG

  *First Bank Selection Data Direction Register.*
- #define BANK0PIN PG3

  *First Bank Selection Pin.*
- #define BANK1PORT PORTG

  *Second Bank Selection Port.*
- #define BANK1DDR DDRG

  *Second Bank Selection Data Direction Register.*
- #define BANK1PIN PG4

  *Second Bank Selection Pin.*
- #define BANK2PORT PORTL

  *Third Bank Selection Port.*
- #define BANK2DDR DDRL

  *Third Bank Selection Data Direction Register.*
- #define BANK2PIN PL5

  *Third Bank Selection Pin.*
- #define SPISS PB0

  *SPI Slave Select Pin.*
- #define RX_BUFFER_SIZE 128

  *UART Receive Buffer Size.*
- #define TX_BUFFER_SIZE 128

  *UART Transmit Buffer Size.*
- #define USB 0

  *FT232RL USB UART module.*
- #define BLUETOOTH 1

  *Bluetooth UART module.*
- #define UART BLUETOOTH

  *UART module to use (0 to 3)*

### 7.3.1 Detailed Description

Various default settings.

Definition in file config.h.

## 7.4 include/debug.h File Reference

Debug and Assert Header and Implementation.

```
#include <avr/wdt.h>
#include <serial.h>
#include <stdio.h>
```

### Macros

- #define DEBUGOUT(x) printf(x)

  *Debug Output Function.*
- #define ASSERTFUNC(x)

  *Simple Assert Implementation.*
- #define assert(x) ASSERTFUNC(x)

  *Enable assert()*
- #define debugPrint(ignore)

  *Disable debugPrint()*

#### 7.4.1 Detailed Description

Debug and Assert Header and Implementation.

Definition in file debug.h.

## 7.5 include/doc.h File Reference

Contains Doxygen Group Definitions.

#### 7.5.1 Detailed Description

Contains Doxygen Group Definitions.

Definition in file doc.h.

## 7.6 include/error.h File Reference

Global listing of different error conditions.

### Macros

- #define CHECKERROR(x) if(x!=SUCCESS){return x;}

  *Check an Error Code.*
- #define REPORTERROR(x)

  *Report an error, if it occured.*

### Enumerations

- enum Error {
  SUCCESS = 0, TWI_NO_ANSWER, TWI_WRITE_ERROR, MALLOC_FAIL,
  ERROR, ARGUMENT_ERROR }

  *Error Conditions.*

### Functions

- char * getErrorString (Error e)

  *Returns a human-readable error description.*

### 7.6.1 Detailed Description

Global listing of different error conditions. Can be returned to signalise error or success. Also allows to print human-readable error descriptions.

Definition in file error.h.

## 7.7 include/gyro.h File Reference

L3GD20 Gyroscope API Header.

```
#include <error.h>
#include <xycontrol.h>
```

### Enumerations

- enum GyroRange { r250DPS, r500DPS, r2000DPS }

  *Gyroscope Range options.*

### Functions

- Error gyroInit (GyroRange r)

  *Initializes the Gyroscope.*
- Error gyroRead (Vector3f *v)

  *Get a set of gyroscope data.*

### 7.7.1 Detailed Description

L3GD20 Gyroscope API Header.

Definition in file gyro.h.

## 7.8 include/kalman.h File Reference

Kalman-Filter Header.

### Data Structures

- struct Kalman

  *Kalman-Filter State data.*

### Functions

- void kalmanInnovate (Kalman *data, double z1, double z2)

  *Step the Kalman Filter.*
- void kalmanInit (Kalman *data)

  *Initialize a Kalman-State.*

### 7.8.1 Detailed Description

Kalman-Filter Header.

Definition in file kalman.h.

## 7.9 include/mag.h File Reference

LSM303DLHC Magnetometer API Header.

```
#include <error.h>
#include <xycontrol.h>
```

### Enumerations

- enum MagRange {
  r1g3 = 1, r1g9 = 2, r2g5 = 3, r4g0 = 4,
  r4g7 = 5, r5g6 = 6, r8g1 = 7 }

  *Magnetometer Range options.*

### Functions

- Error magInit (MagRange r)

  *Initialize the Magnetometer.*
- Error magRead (Vector3f *v)

  *Read from the Magnetometer.*

### 7.9.1 Detailed Description

LSM303DLHC Magnetometer API Header.

Definition in file mag.h.

## 7.10 include/motor.h File Reference

BL-Ctrl V1.2 Controller API Header.

```
#include <config.h>
```

### Functions

- void motorInit (void)

  *Initializes the motor control library.*
- void motorSet (uint8_t id, uint8_t speed)

  *Set the speed of one or all motors.*
- void motorTask (void)

  *Send the values stored in motorSpeed to the Controllers.*

### Variables

- uint8_t motorSpeed [MOTORCOUNT]
  *Speed for the four motors.*

#### 7.10.1 Detailed Description

BL-Ctrl V1.2 Controller API Header.

Definition in file motor.h.

## 7.11 include/orientation.h File Reference

Orientation API Header.

```
#include <error.h>
```

### Data Structures

- struct Angles
  *Can store orientation in Euler Space.*

### Functions

- Error orientationInit (void)
  *Initializes the Orientation API.*
- Error orientationTask (void)
  *Calculate the current orientation.*
- void zeroOrientation (void)
  *Sets the current orientation to zero.*

### Variables

- Angles orientation
  *Current Aircraft orientation.*

#### 7.11.1 Detailed Description

Orientation API Header.

Definition in file orientation.h.

## 7.12 include/pid.h File Reference

PID Library Header.

### Data Structures

- struct PIDState
  *Data Structure for a single PID Controller.*

### Macros

- #define ROLL 0

  *Roll index for o_should, o_output and o_pids.*
- #define PITCH 1

  *Pitch index for o_should, o_output and o_pids.*

### Functions

- void pidInit (void)

  *Initialize Roll and Pitch PID.*
- void pidTask (void)

  *Step the Roll and Pitch PID Controllers.*
- void pidSet (PIDState *pid, double kp, double ki, double kd, double min, double max, double iMin, double iMax)

  *Set the parameters of a PID controller.*
- double pidExecute (double should, double is, PIDState *state)

  *Execute a single PID Control Step.*

### Variables

- double o_should [2]

  *Roll and Pitch target angles.*
- double o_output [2]

  *Roll and Pitch PID Output.*
- PIDState o_pids [2]

  *Roll and Pitch PID States.*

#### 7.12.1 Detailed Description

PID Library Header.

Definition in file pid.h.

## 7.13 include/serial.h File Reference

UART API Header.

### Macros

- #define BAUD(baudRate, xtalCpu) ((xtalCpu)/((baudRate)*16l)-1)

  *Calculate Baudrate Register Value.*

### Functions

- void serialInit (uint16_t baud)

  *Initialize the UART Hardware.*
- void serialClose (void)

  *Stop the UART Hardware.*
- void setFlow (uint8_t on)

*Manually change the flow control.*

- uint8_t serialHasChar (void)

  *Check if a byte was received.*
- uint8_t serialGet (void)

  *Read a single byte.*
- uint8_t serialGetBlocking (void)

  *Wait until a character is received.*
- uint8_t serialRxBufferFull (void)

  *Check if the receive buffer is full.*
- uint8_t serialRxBufferEmpty (void)

  *Check if the receive buffer is empty.*
- void serialWrite (uint8_t data)

  *Send a byte.*
- void serialWriteString (const char *data)

  *Send a string.*
- uint8_t serialTxBufferFull (void)

  *Check if the transmit buffer is full.*
- uint8_t serialTxBufferEmpty (void)

  *Check if the transmit buffer is empty.*

### 7.13.1 Detailed Description

UART API Header.

Definition in file serial.h.

## 7.14 include/set.h File Reference

Motor Mixer Library Header.

### Functions

- void setTask (void)

  *Read the PID Output and Set the Motor Speeds.*

### Variables

- uint8_t baseSpeed

  *Motor Base Speed.*

### 7.14.1 Detailed Description

Motor Mixer Library Header.

Definition in file set.h.

## 7.15 include/spi.h File Reference

SPI API Header.

### Enumerations

- enum SPI_MODE { MODE_0 = 0, MODE_1 = 1, MODE_2 = 2, MODE_3 = 3 }

  *SPI Mode option.*
- enum SPI_SPEED {
  SPEED_2 = 4, SPEED_4 = 0, SPEED_8 = 5, SPEED_16 = 1,
  SPEED_32 = 6, SPEED_64 = 2, SPEED_128 = 3 }

  *SPI Speed options.*

### Functions

- void spiInit (SPI_MODE mode, SPI_SPEED speed)

  *Initialize the SPI Hardware Module.*
- uint8_t spiSendByte (uint8_t d)

  *Send and Receive one byte.*

#### 7.15.1 Detailed Description

SPI API Header.

Definition in file spi.h.

## 7.16 include/tasks.h File Reference

Task API Header.

### Data Structures

- struct TaskElement

  *Single-Linked Task List.*

### Typedefs

- typedef void(∗ Task )(void)

  *A Task has no arguments and returns nothing.*

### Functions

- uint8_t addTask (Task func)

  *Adds another task that will be called regularly.*
- uint8_t removeTask (Task func)

  *Removes an already registered Task.*
- void tasks (void)

  *Executes registered Tasks.*
- uint8_t tasksRegistered (void)

  *Get the number of registered Tasks.*

### Variables

- TaskElement * taskList

  *List of registered Tasks.*

#### 7.16.1 Detailed Description

Task API Header.

Definition in file tasks.h.

## 7.17 include/time.h File Reference

Time API Header.

### Typedefs

- typedef uint64_t time_t

  *Timekeeping Data Type.*

### Functions

- void initSystemTimer (void)

  *Initialize the system timer.*
- time_t getSystemTime (void)

  *Get the System Uptime.*

#### 7.17.1 Detailed Description

Time API Header.

Definition in file time.h.

## 7.18 include/twi.h File Reference

I2C API Header.

### Macros

- #define TWI_READ 1

  *I2C Read Bit.*
- #define TWI_WRITE 0

  *I2C Write Bit.*

### Functions

- void twiInit (void)

  *Initialize the I2C Hardware.*
- void twiStop (void)

  *Stop the I2C Hardware.*
- unsigned char twiStart (unsigned char addr)

  *Start an I2C Transfer.*
- unsigned char twiRepStart (unsigned char addr)

  *Start a repeated I2C Transfer.*
- void twiStartWait (unsigned char addr)

  *Start an I2C Transfer and poll until ready.*
- unsigned char twiWrite (unsigned char data)

  *Write to the I2C Slave.*
- unsigned char twiReadAck (void)

  *Read from the I2C Slave and request more data.*
- unsigned char twiReadNak (void)

  *Read from the I2C Slave and deny more data.*

#### 7.18.1 Detailed Description

I2C API Header.

Definition in file twi.h.

## 7.19 include/uartMenu.h File Reference

UART Menu API Header.

```
#include <tasks.h>
```

### Data Structures

- struct MenuEntry

  *Data Structure for Single-Linked-List for UART Menu.*

### Functions

- uint8_t addMenuCommand (uint8_t cmd, PGM_P help, Task f)

  *Add a command to the UART Menu.*
- void uartMenuPrintHelp (void)

  *Print all registered commands.*
- void uartMenuRegisterHandler (void(∗handler)(char))

  *Register a Handler for unhandled menu commands.*
- void uartMenuTask (void)

  *Task to work the UART Menu.*

### 7.19.1 Detailed Description

UART Menu API Header.

Definition in file uartMenu.h.

## 7.20 include/xmem.h File Reference

XMEM API Header.

### Data Structures

- struct MallocState

  *All Malloc related State.*

### Macros

- #define MEMSWITCH(x) uint8_t oldMemBank=xmemGetBank();if(oldMemBank!=x)xmemSetBank(x);

  *Switch the bank, if needed.*
- #define MEMSWITCHBACK(x) if(oldMemBank!=x)xmemSetBank(oldMemBank);

  *Switch back to the last bank, if needed.*
- #define MEMBANKS 8

  *Available Memory Banks.*
- #define BANK_GENERIC 0

  *Generic Memory Bank.*

### Functions

- void xmemInit (void)

  *Initialize the External Memory Interface.*
- void xmemSetBank (uint8_t bank)

  *Switch the active memory bank.*
- uint8_t xmemGetBank (void)

  *Get the current memory bank.*

### Variables

- MallocState states [MEMBANKS]

  *MallocState for all Memory Banks.*
- uint8_t currentBank

  *Current active Memory Bank.*

### 7.20.1 Detailed Description

XMEM API Header.

Definition in file xmem.h.

## 7.21 include/xycontrol.h File Reference

xyControl API Header.

### Data Structures

- struct Vector3f

  *The global 3-Dimensional Floating Point Vector.*

### Enumerations

- enum LED {
  LED_RED0 = 0, LED_RED1 = 1, LED_GREEN0 = 2, LED_GREEN1 = 3,
  LED_ALL = 4, LED_BITMAP = 5, LED_RED = 6, LED_GREEN = 7 }

  *Methods of addressing the LEDs.*
- enum LEDState { LED_OFF = 0, LED_ON = 1, LED_TOGGLE = 2 }

  *Possible states of the LEDs.*

### Functions

- void xyInit (void)

  *Initialize the xyControl Hardware.*
- void xyLed (LED l, LEDState v)

  *Set the LEDs.*
- double getVoltage (void)

  *Calculate and return the Battery Voltage.*
- void resetSelf (void)

  *Use the Watchdog to reset yourself after 15ms.*

#### 7.21.1 Detailed Description

xyControl API Header.

Definition in file xycontrol.h.

## 7.22 lib/acc.c File Reference

LSM303DLHC Accelerometer API Implementation.

```
#include <avr/io.h>
#include <stdint.h>
#include <stdlib.h>
#include <twi.h>
#include <acc.h>
#include <error.h>
#include <config.h>
```

### Macros

- #define ACCREG_CTRL1 0x20

  *Accelerometer Control Register 1.*
- #define ACCREG_CTRL4 0x23

  *Accelerometer Control Register 4.*
- #define ACCREG_XL 0x28

  *First Accelerometer Output Register.*

### Functions

- Error accWriteRegister (uint8_t reg, uint8_t val)

  *Write an Accelerometer Register.*
- Error accInit (AccRange r)

  *Initialize the Accelerometer.*
- Error accRead (Vector3f *v)

  *Read from the Accelerometer.*

### Variables

- AccRange accRange

  *Stored range to scale returned values.*

#### 7.22.1 Detailed Description

LSM303DLHC Accelerometer API Implementation.

Definition in file acc.c.

## 7.23 lib/adc.c File Reference

Analog-to-Digital Converter API Implementation.

```
#include <avr/io.h>
#include <stdint.h>
#include <adc.h>
```

### Functions

- void adcInit (ADCRef ref)

  *Initialize the ADC Hardware.*
- void adcStart (uint8_t channel)

  *Start a conversion on a given channel.*
- uint8_t adcReady (void)

  *Check if a result is ready.*
- uint16_t adcGet (uint8_t next)

  *Get the conversion results.*
- void adcClose (void)

  *Disable the ADC to save energy.*

### 7.23.1 Detailed Description

Analog-to-Digital Converter API Implementation.

Definition in file adc.c.

## 7.24 lib/error.c File Reference

Global listing of different error conditions.

```
#include <avr/io.h>
#include <stdint.h>
#include <stdlib.h>
#include <avr/pgmspace.h>
#include <error.h>
```

### Functions

- char * getErrorString (Error e)

  *Returns a human-readable error description.*

### Variables

- char PROGMEM error0 [] = "Success"

  *String for SUCCESS.*
- char PROGMEM error1 [] = "TWI doesn't answer"

  *String for TWI_NO_ANSWER.*
- char PROGMEM error2 [] = "TWI could not write"

  *String for TWI_WRITE_ERROR.*
- char PROGMEM error3 [] = "Not enough memory"

  *String for MALLOC_FAIL.*
- char PROGMEM error4 [] = "General Error"

  *String for ERROR.*
- char PROGMEM error5 [] = "Argument Error"

  *String for ARGUMENT_ERROR.*
- PGM_P PROGMEM errorTable []

  *Array of all error descriptions in Flash Memory.*

### 7.24.1 Detailed Description

Global listing of different error conditions. Can be returned to signalise error or success. Also allows to print human-readable error descriptions.

Definition in file error.c.

### 7.24.2 Variable Documentation

#### 7.24.2.1 char PROGMEM error0[ ] = "Success"

String for SUCCESS.

Definition at line 43 of file error.c.

#### 7.24.2.2 char PROGMEM error1[ ] = "TWI doesn't answer"

String for TWI_NO_ANSWER.

Definition at line 44 of file error.c.

#### 7.24.2.3 char PROGMEM error2[ ] = "TWI could not write"

String for TWI_WRITE_ERROR.

Definition at line 45 of file error.c.

#### 7.24.2.4 char PROGMEM error3[ ] = "Not enough memory"

String for MALLOC_FAIL.

Definition at line 46 of file error.c.

#### 7.24.2.5 char PROGMEM error4[ ] = "General Error"

String for ERROR.

Definition at line 47 of file error.c.

#### 7.24.2.6 char PROGMEM error5[ ] = "Argument Error"

String for ARGUMENT_ERROR.

Definition at line 48 of file error.c.

#### 7.24.2.7 PGM_P PROGMEM errorTable[ ]

**Initial value:**

```
= {
    error0, error1, error2, error3, error4, error5
}
```

Array of all error descriptions in Flash Memory.

Definition at line 51 of file error.c.

Referenced by getErrorString().

## 7.25 lib/gyro.c File Reference

L3GD20 Gyroscope API Implementation.

```
#include <stdlib.h>
#include <stdint.h>
#include <avr/io.h>
#include <twi.h>
#include <gyro.h>
#include <error.h>
#include <config.h>
```

### Macros

- #define GYROREG_CTRL1 0x20

  *Gyroscope Control Register 1.*
- #define GYROREG_CTRL4 0x23

  *Gyroscope Control Register 4.*
- #define GYROREG_OUTXL 0x28

  *First Gyroscope Output Register.*

### Functions

- Error gyroWriteByte (uint8_t reg, uint8_t val)

  *Write a Gyroscope Register.*
- Error gyroInit (GyroRange r)

  *Initializes the Gyroscope.*
- Error gyroRead (Vector3f *v)

  *Get a set of gyroscope data.*

### Variables

- GyroRange gyroRange

  *Stored range to scale returned values.*

#### 7.25.1 Detailed Description

L3GD20 Gyroscope API Implementation.

Definition in file gyro.c.

## 7.26 lib/kalman.c File Reference

Kalman-Filter Implementation.

```
#include <stdint.h>
#include <avr/io.h>
#include <kalman.h>
#include <config.h>
```

### Functions

- void kalmanInit (Kalman *data)

  *Initialize a Kalman-State.*
- void kalmanInnovate (Kalman *data, double z1, double z2)

  *Step the Kalman Filter.*

#### 7.26.1 Detailed Description

Kalman-Filter Implementation.

Definition in file kalman.c.

## 7.27 lib/mag.c File Reference

LSM303DLHC Magnetometer API Implementation.

```
#include <avr/io.h>
#include <stdint.h>
#include <stdlib.h>
#include <twi.h>
#include <mag.h>
#include <error.h>
#include <config.h>
```

### Macros

- #define MAGREG_CRB 0x01

  *Magnetometer Gain Register.*
- #define MAGREG_MR 0x02

  *Magnetometer Mode Register.*
- #define MAGREG_XH 0x03

  *First Magnetometer Output Register.*

### Functions

- Error magWriteRegister (uint8_t reg, uint8_t val)

  *Write a Magnetometer Register.*
- Error magInit (MagRange r)

  *Initialize the Magnetometer.*
- Error magRead (Vector3f *v)

  *Read from the Magnetometer.*

### 7.27.1 Detailed Description

LSM303DLHC Magnetometer API Implementation.

Definition in file mag.c.

## 7.28 lib/motor.c File Reference

BL-Ctrl V1.2 Controller API Implementation.

```
#include <stdint.h>
#include <avr/io.h>
#include <twi.h>
#include <motor.h>
#include <tasks.h>
#include <time.h>
#include <config.h>
```

### Functions

- void motorTask (void)

  *Send the values stored in motorSpeed to the Controllers.*
- void motorInit (void)

  *Initializes the motor control library.*
- void motorSet (uint8_t id, uint8_t speed)

  *Set the speed of one or all motors.*

### Variables

- uint8_t motorSpeed [MOTORCOUNT]

  *Speed for the four motors.*

#### 7.28.1 Detailed Description

BL-Ctrl V1.2 Controller API Implementation.

Definition in file motor.c.

## 7.29 lib/orientation.c File Reference

Orientation API Implementation.

```
#include <avr/io.h>
#include <stdint.h>
#include <math.h>
#include <xycontrol.h>
#include <error.h>
#include <gyro.h>
#include <acc.h>
#include <mag.h>
#include <tasks.h>
#include <time.h>
#include <orientation.h>
#include <kalman.h>
#include <config.h>
```

### Macros

- #define TODEG(x) ((x * 180) / M_PI)

  *Convert Radians to Degrees.*

### Functions

- Error orientationInit (void)

  *Initializes the Orientation API.*
- Error orientationTask (void)

  *Calculate the current orientation.*
- void zeroOrientation (void)

  *Sets the current orientation to zero.*

### Variables

- Angles orientation = {.pitch = 0, .roll = 0, .yaw = 0}

  *Current Aircraft orientation.*
- Angles orientationError = {.pitch = 0, .roll = 0, .yaw = 0}

  *Current Aircraft orientation offset.*
- Kalman pitchData

  *Kalman-State for Pitch Angle.*
- Kalman rollData

  *Kalman-State for Roll Angle.*

#### 7.29.1 Detailed Description

Orientation API Implementation.

Definition in file orientation.c.

## 7.30 lib/pid.c File Reference

PID Library Implementation.

```
#include <stdint.h>
#include <avr/io.h>
#include <twi.h>
#include <motor.h>
#include <tasks.h>
#include <time.h>
#include <pid.h>
#include <orientation.h>
#include <config.h>
```

### Functions

- double pidExecute (double should, double is, PIDState *state)

  *Execute a single PID Control Step.*
- void pidInit (void)

  *Initialize Roll and Pitch PID.*
- void pidSet (PIDState *pid, double kp, double ki, double kd, double min, double max, double iMin, double iMax)

  *Set the parameters of a PID controller.*
- void pidTask (void)

  *Step the Roll and Pitch PID Controllers.*

### Variables

- PIDState o_pids [2]

  *Roll and Pitch PID States.*
- double o_should [2]

  *Roll and Pitch target angles.*
- double o_output [2]

  *Roll and Pitch PID Output.*

### 7.30.1 Detailed Description

PID Library Implementation.

Definition in file pid.c.

## 7.31 lib/serial.c File Reference

UART API Implementation.

```
#include <avr/io.h>
#include <avr/interrupt.h>
#include <stdint.h>
#include <serial.h>
#include <config.h>
```

### Macros

- #define XON 0x11

  *XON Byte.*
- #define XOFF 0x13

  *XOFF Byte.*
- #define FLOWMARK 5

  *Space remaining to trigger XOFF/XON.*

### Functions

- ISR (SERIALRECIEVEINTERRUPT)

  *Receive Complete Interrupt.*
- ISR (SERIALTRANSMITINTERRUPT)

  *Data Register Empty Interrupt.*
- void serialInit (uint16_t baud)

  *Initialize the UART Hardware.*
- void serialClose (void)

  *Stop the UART Hardware.*
- void setFlow (uint8_t on)

  *Manually change the flow control.*
- uint8_t serialHasChar (void)

  *Check if a byte was received.*
- uint8_t serialGetBlocking (void)

  *Wait until a character is received.*
- uint8_t serialGet (void)

  *Read a single byte.*
- uint8_t serialRxBufferFull (void)

  *Check if the receive buffer is full.*
- uint8_t serialRxBufferEmpty (void)

  *Check if the receive buffer is empty.*
- void serialWrite (uint8_t data)

  *Send a byte.*
- void serialWriteString (const char *data)

*Send a string.*

- uint8_t serialTxBufferFull (void)

  *Check if the transmit buffer is full.*

- uint8_t serialTxBufferEmpty (void)

  *Check if the transmit buffer is empty.*

### Variables

- uint8_t volatile rxBuffer [RX_BUFFER_SIZE]

  *RX FIFO Buffer.*

- uint8_t volatile txBuffer [TX_BUFFER_SIZE]

  *TX FIFO Buffer.*

- uint16_t volatile rxRead = 0

  *RX FIFO Read Position.*

- uint16_t volatile rxWrite = 0

  *RX FIFO Write Position.*

- uint16_t volatile txRead = 0

  *TX FIFO Read Position.*

- uint16_t volatile txWrite = 0

  *TX FIFO Write Position.*

- uint8_t volatile shouldStartTransmission = 1

  *Should enable interrupt.*

#### 7.31.1 Detailed Description

UART API Implementation.

Definition in file serial.c.

## 7.32 lib/set.c File Reference

Motor Mixer Library Implementation.

```
#include <stdint.h>
#include <avr/io.h>
#include <twi.h>
#include <motor.h>
#include <tasks.h>
#include <time.h>
#include <pid.h>
#include <set.h>
#include <config.h>
```

### Macros

- #define MAXDIFF (baseSpeed * PID_FACTOR)

  *Maximum Speed difference on one axis.*

### Functions

- void setMotorSpeeds (uint8_t axis, uint8_t *vals)

  *Set the Motor Speeds according to the SET_* Motor Position Constants.*
- void setTask (void)

  *Read the PID Output and Set the Motor Speeds.*

### Variables

- uint8_t baseSpeed = 0

  *Motor Base Speed.*

#### 7.32.1 Detailed Description

Motor Mixer Library Implementation.

Definition in file set.c.

## 7.33 lib/spi.c File Reference

SPI API Implementation.

```
#include <stdint.h>
#include <avr/io.h>
#include <spi.h>
#include <config.h>
```

### Functions

- uint8_t spiSendByte (uint8_t d)

  *Send and Receive one byte.*

#### 7.33.1 Detailed Description

SPI API Implementation.

Definition in file spi.c.

## 7.34 lib/tasks.c File Reference

Task API Implementation.

```
#include <stdlib.h>
#include <stdint.h>
#include <xmem.h>
#include <tasks.h>
```

### Functions

- uint8_t tasksRegistered (void)

  *Get the number of registered Tasks.*
- uint8_t addTask (Task func)

  *Adds another task that will be called regularly.*
- uint8_t removeTask (Task func)

  *Removes an already registered Task.*
- void tasks (void)

  *Executes registered Tasks.*

### Variables

- TaskElement * taskList = NULL

  *List of registered Tasks.*

#### 7.34.1 Detailed Description

Task API Implementation.

Definition in file tasks.c.

## 7.35 lib/time.c File Reference

Time API Implementation.

```
#include <stdlib.h>
#include <stdint.h>
#include <avr/io.h>
#include <avr/interrupt.h>
#include <util/atomic.h>
#include <time.h>
```

### Macros

- #define TCRA TCCR2A

  *Timer 2 Control Register A.*
- #define TCRB TCCR2B

  *Timer 2 Control Register B.*
- #define OCR OCR2A

  *Timer 2 Compare Register A.*
- #define TIMS TIMSK2

  *Timer 2 Interrupt Mask.*
- #define OCIE OCIE2A

  *Timer 2 Compare Match A Interrupt Enable.*

### Functions

- void initSystemTimer (void)

  *Initialize the system timer.*
- ISR (TIMER2_COMPA_vect)

  *Timer 2 Compare Match A Interrupt.*
- time_t getSystemTime (void)

  *Get the System Uptime.*

### Variables

- volatile time_t systemTime = 0

  *Current System Uptime.*

#### 7.35.1 Detailed Description

Time API Implementation.

Definition in file time.c.

## 7.36 lib/uartMenu.c File Reference

UART Menu API Implementation.

```
#include <avr/io.h>
#include <stdint.h>
#include <stdlib.h>
#include <stdio.h>
#include <avr/pgmspace.h>
#include <xycontrol.h>
#include <xmem.h>
#include <tasks.h>
#include <serial.h>
#include <uartMenu.h>
```

### Functions

- MenuEntry * findEntry (uint8_t cmd)

  *Search the uartMenu Linked List.*
- uint8_t addMenuCommand (uint8_t cmd, PGM_P help, Task f)

  *Add a command to the UART Menu.*
- MenuEntry * reverseList (MenuEntry *root)

  *Reverse the UART Menu List.*
- void uartMenuPrintHelp (void)

  *Print all registered commands.*
- void uartMenuRegisterHandler (void(*handler)(char))

  *Register a Handler for unhandled menu commands.*
- void uartMenuTask (void)

  *Task to work the UART Menu.*

### Variables

- MenuEntry * uartMenu = NULL
  *Single-Linked-List for commands.*
- void(* unHandler )(char) = NULL
  *Handler for unhandled commands.*

### 7.36.1 Detailed Description

UART Menu API Implementation.

Definition in file uartMenu.c.

## 7.37 lib/xmem.c File Reference

XMEM API Implementation.

```
#include <avr/io.h>
#include <stdint.h>
#include <stdlib.h>
#include <xmem.h>
#include <config.h>
```

### Functions

- void saveState (uint8_t bank)
  *Save the current malloc state.*
- void restoreState (uint8_t bank)
  *Restore the malloc state.*
- void xmemInit (void)
  *Initialize the External Memory Interface.*
- void xmemSetBank (uint8_t bank)
  *Switch the active memory bank.*
- uint8_t xmemGetBank (void)
  *Get the current memory bank.*

### Variables

- MallocState states [MEMBANKS]
  *MallocState for all Memory Banks.*
- uint8_t currentBank = 0
  *Current active Memory Bank.*
- void * __brkval
  *Internal Malloc Heap-End Pointer.*
- void * __flp
  *Internal Malloc Free List Pointer (State)*

### 7.37.1 Detailed Description

XMEM API Implementation.

Definition in file xmem.c.

## 7.38 lib/xycontrol.c File Reference

xyControl API Implementation.

```
#include <avr/io.h>
#include <stdint.h>
#include <stdlib.h>
#include <stdio.h>
#include <avr/interrupt.h>
#include <avr/pgmspace.h>
#include <avr/wdt.h>
#include <serial.h>
#include <spi.h>
#include <time.h>
#include <xmem.h>
#include <xycontrol.h>
#include <twi.h>
#include <adc.h>
#include <uartMenu.h>
#include <tasks.h>
#include <config.h>
```

### Functions

- int uartoutput (char c, FILE *f)

  *Method used to write to stdout and stderr.*
- int uartinput (FILE *f)

  *Method used to read from stdin.*
- void xyInit (void)

  *Initialize the xyControl Hardware.*
- void xyLedInternal (uint8_t v, volatile uint8_t *port, uint8_t pin)

  *Internal LED Manipulation function.*
- double getVoltage (void)

  *Calculate and return the Battery Voltage.*
- void resetSelf (void)

  *Use the Watchdog to reset yourself after 15ms.*

### Variables

- char PROGMEM helpText [] = "Print this Help"

  *UART Menu Help Text.*
- char PROGMEM resetText [] = "Reset MCU"

  *UART Menu Reset Text.*
- FILE inFile

  *FILE for stdin.*
- FILE outFile

  *FILE for stdout and stderr.*

### 7.38.1 Detailed Description

xyControl API Implementation.

Definition in file xycontrol.c.

# Chapter 8

# Example Documentation

## 8.1 uartFlight.c

```
/*
 * uartFlight.c
 *
 * Copyright (c) 2013, Thomas Buck <xythobuz@me.com>
 * All rights reserved.
 *
 * Redistribution and use in source and binary forms, with or without
 * modification, are permitted provided that the following conditions
 * are met:
 *
 * - Redistributions of source code must retain the above copyright notice,
 *   this list of conditions and the following disclaimer.
 *
 * - Redistributions in binary form must reproduce the above copyright
 *   notice, this list of conditions and the following disclaimer in the
 *   documentation and/or other materials provided with the distribution.
 *
 * THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS
 * "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED
 * TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR
 * PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT HOLDER OR
 * CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL,
 * EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO,
 * PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR
 * PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF
 * LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING
 * NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS
 * SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
 */
#include <stdint.h>
#include <stdlib.h>
#include <stdio.h>
#include <avr/io.h>
#include <avr/pgmspace.h>

#include <tasks.h>
#include <error.h>
#include <xycontrol.h>
#include <time.h>
#include <uartMenu.h>
#include <serial.h>
#include <acc.h>
#include <gyro.h>
#include <mag.h>
#include <motor.h>
#include <orientation.h>
#include <pid.h>
#include <set.h>

#define MAXANGLE 45
#define ANGLESTEP 10
#define MAXMOTOR 255
#define MOTORSTEP 10
#define QUADFREQ 100
#define STATUSFREQ 10

#define QUADDELAY (1000 / QUADFREQ)
#define STATUSDELAY (1000 / STATUSFREQ)
```

```
void flightTask(void);
void statusTask(void);
void controlToggle(void);
void motorToggle(void);
void motorUp(void);
void motorDown(void);
void motorForward(void);
void motorBackward(void);
void motorLeft(void);
void motorRight(void);
void parameterChange(void);

char PROGMEM motorToggleString[] = "Motor On/Off";
char PROGMEM motorUpString[] = "Up";
char PROGMEM motorDownString[] = "Down";
char PROGMEM motorLeftString[] = "Left";
char PROGMEM motorRightString[] = "Right";
char PROGMEM motorForwardString[] = "Forwards";
char PROGMEM motorBackwardString[] = "Backwards";
char PROGMEM controlToggleString[] = "Toggle PID";
char PROGMEM parameterChangeString[] = "Change PID Params";
char PROGMEM zeroString[] = "Angles to Zero";

uint8_t state = 0; // Bit 0: Motor, Bit 1: PID
uint8_t speed = 10;
int16_t targetRoll = 0;
int16_t targetPitch = 0;

uint32_t sumFlightTask = 0, sumFlightCount = 0;

int main(void) {
    xyInit();
    pidInit();
    motorInit();
    orientationInit();

    addTask(&flightTask);
    addTask(&statusTask);

    addMenuCommand('m', motorToggleString, &motorToggle);
    addMenuCommand('w', motorForwardString, &motorForward);
    addMenuCommand('a', motorLeftString, &motorLeft);
    addMenuCommand('s', motorBackwardString, &motorBackward);
    addMenuCommand('d', motorRightString, &motorRight);
    addMenuCommand('x', motorUpString, &motorUp);
    addMenuCommand('y', motorDownString, &motorDown);
    addMenuCommand('p', controlToggleString, &controlToggle);
    addMenuCommand('n', parameterChangeString, &parameterChange);
    addMenuCommand('z', zeroString, &zeroOrientation);

    xyLed(LED_ALL, LED_ON);

    for(;;) {
        tasks();
    }

    return 0;
}

void flightTask(void) {
    static time_t last = 100; // Don't begin immediately
    if ((getSystemTime() - last) >= QUADDELAY) {
        last = getSystemTime();
        Error e = orientationTask();
        REPORTERROR(e);
        if (state & 0x02) {
            pidTask();
        } else {
            o_output[0] = o_output[1] = 0;
        }
        setTask();
        motorTask();

        uint32_t diff = getSystemTime() - last;
        if (++sumFlightCount >= QUADFREQ) {
            sumFlightCount = 1;
            sumFlightTask = diff;
        } else {
            sumFlightTask += diff;
        }
    }
}

void statusTask(void) {
    static time_t last = 100; // Don't begin immediately
    static uint32_t lastDuration = 0;
    if ((getSystemTime() - last) >= STATUSDELAY) {
```

```
        last = getSystemTime();
        printf("q%li %li\n", sumFlightTask / sumFlightCount, lastDuration);
        printf("r%.2f %.2f\n", o_pids[0].intMin, o_pids[0].intMax);
        printf("s%.2f %.2f\n", o_pids[0].outMin, o_pids[0].outMax);
        printf("t%.3f %.3f %.3f\n", o_pids[0].kp, o_pids[0].ki,
      o_pids[0].kd);
        printf("u%.2f %.2f\n", o_output[PITCH], o_output[
      ROLL]);
        printf("v%i %i %i %i\n", motorSpeed[0], motorSpeed[1],
      motorSpeed[2], motorSpeed[3]);
        printf("w%.2f\n", orientation.pitch);
        printf("x%.2f\n", orientation.roll);
        printf("y%.2f\n", orientation.yaw);
        printf("z%.2f\n", getVoltage());
        lastDuration = getSystemTime() - last;
    }
}

void controlToggle(void) {
    if (state & 0x02) {
        state &= ~0x02;
        printf("PID Off!\n");
    } else {
        state |= 0x02;
        printf("PID On!\n");
    }
}

void motorToggle(void) {
    if (state & 0x01) {
        state &= ~0x01;
        baseSpeed = 0;
        printf("Motor Off!\n");
    } else {
        state |= 0x01;
        baseSpeed = speed = 10;
        printf("Motor On!\n");
    }
}

void motorUp(void) {
    if (speed <= (MAXMOTOR - MOTORSTEP)) {
        if (state & 0x01) {
            speed += MOTORSTEP;
            baseSpeed = speed;
            printf("Throttle up to %i\n", speed);
        }
    }
}

void motorDown(void) {
    if (speed >= MOTORSTEP) {
        if (state & 0x01) {
            speed -= MOTORSTEP;
            baseSpeed = speed;
            printf("Throttle down to %i\n", speed);
        }
    }
}

void motorForward(void) {
    if (targetPitch >= (-1 * (MAXANGLE - ANGLESTEP))) {
        targetPitch -= ANGLESTEP;
        o_should[PITCH] = targetPitch;
        printf("Pitch Forward %i\n", targetPitch);
    }
}

void motorBackward(void) {
    if (targetPitch <= (MAXANGLE - ANGLESTEP)) {
        targetPitch += ANGLESTEP;
        o_should[PITCH] = targetPitch;
        printf("Pitch Backwards %i\n", targetPitch);
    }
}

void motorLeft(void) {
    if (targetRoll <= (MAXANGLE - ANGLESTEP)) {
        targetRoll += ANGLESTEP;
        o_should[ROLL] = targetRoll;
        printf("Roll Left %i\n", targetRoll);
    }
}

void motorRight(void) {
    if (targetRoll >= (-1 * (MAXANGLE - ANGLESTEP))) {
        targetRoll -= ANGLESTEP;
```

```
        o_should[ROLL] = targetRoll;
        printf("Roll Right %i\n", targetRoll);
    }
}

void parameterChange(void) {
    double p, i, d, min, max, iMin, iMax;
    int c = scanf("%lf %lf %lf %lf %lf %lf %lf", &p, &i, &d, &min, &max, &iMin, &iMax);
    if (c == 7) {
        pidSet(&o_pids[0], p, i, d, min, max, iMin, iMax);
        pidSet(&o_pids[1], p, i, d, min, max, iMin, iMax);
    } else {
        printf("Only got %i (%lf %lf %lf %lf %lf %lf %lf)!\n", c, p, i, d, min, max, iMin, iMax);
    }
}
```

# Index