xyControl

0.1

Generated by Doxygen 1.8.3.1

Sun Jul 7 2013 18:19:25

Contents

Chapter 1

Main Page

`xyControl` is a Quadrocopter Flight Controller based on Atmels Atmega2560 microcontroller. It features 512KB 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.

`![Photo 1][xy1s] ![Photo 2][xy2s]`

Software used

- [Peter Fleurys TWI Library][fleury]

License

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

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

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.

5.1.1 Detailed Description

Software Libraries.

5.2 System

System Libraries.

Modules

- Datatypes
- Error Reporting

  Error reporting with human readable strings.
- Time Keeping

  Measuring Time with Millisecond Resolution.
- External Memory Interface

  Allows access to external RAM with bank-switching.
- Remote Control Interface

  Read RC Receiver Sum Signal.

5.2.1 Detailed Description

System Libraries.

5.3 Hardware

Hardware Libraries.

Modules

- Accelerometer Driver

  Configuring and reading an LSM303DLHC Accelerometer.
- Gyroscope Driver

  Configuring and reading an L3GD20.
- ADC Driver

  Analog-to-Digital Converter Library.
- UART Library

  UART Library enabling you to control all available UART Modules.
- I2C Driver

  Using the AVR TWI/I2C Hardware.
- Magnetometer Driver

  Configuring and reading an LSM303DLHC Magnetometer.

5.3.1 Detailed Description

Hardware Libraries.

5.4 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 ACC_ADDRESS 0x32

  Accelerometer Address (0011001r)
- #define ACCFILTERFACTOR 1

  Accelerometer Low Pass Factor.
- #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.

5.4.1 Detailed Description

Configuring and reading an LSM303DLHC Accelerometer.

5.4.2 Macro Definition Documentation

5.4.2.1 #define ACC_ADDRESS 0x32

Accelerometer Address (0011001r)

Definition at line 46 of file acc.h.

Referenced by accRead().

5.4.2.2 #define ACCFILTERFACTOR 1

Accelerometer Low Pass Factor.

Definition at line 49 of file acc.h.

Referenced by accRead().

5.4.2.3 #define ACCREG_CTRL1 0x20

Accelerometer Control Register 1.

Definition at line 47 of file acc.c.

Referenced by accInit().

5.4.2.4 #define ACCREG_CTRL4 0x23

Accelerometer Control Register 4.

Definition at line 48 of file acc.c.

Referenced by accInit().

5.4.2.5 #define ACCREG_XL 0x28

First Accelerometer Output Register.

Definition at line 49 of file acc.c.

Referenced by accRead().

5.4.3 Enumeration Type Documentation

5.4.3.1 enum AccRange

Accelerometer Range options.

Enumerator

r2G +- 2G

r4G +- 4G

r8G +- 8G

r16G +- 16G

Definition at line 52 of file acc.h.

```
             {
    r2G,
    r4G,
    r8G,
    r16G,
} AccRange;
```

5.4.4 Function Documentation

5.4.4.1 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.

Examples:

flight.c.

Definition at line 74 of file acc.c.

References accRange, ACCREG_CTRL1, ACCREG_CTRL4, accWriteRegister(), ARGUMENT_ERROR, r16G, r2-G, r4G, r8G, and SUCCESS.

```
74                            {
75     uint8_t v;
76     switch (r) {
77         case r2G:
78             v = 0x00;
79             break;
80         case r4G:
81             v = 0x10;
82             break;
83         case r8G:
84             v = 0x20;
85             break;
86         case r16G:
87             v = 0x30;
88             break;
89         default:
90             return ARGUMENT_ERROR;
91     }
92     accRange = r;
93     Error e = accWriteRegister(ACCREG_CTRL1, 0x57); // Enable all axes,
       100Hz
94     if (e != SUCCESS) {
95         return e;
96     }
97     e = accWriteRegister(ACCREG_CTRL4, v);
98     return e;
99 }
```

5.4.4.2 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 101 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.

```
101                                    {
102         static double accSumX = 0; /* Buffer for X Low-Pass. */
103         static double accSumY = 0; /* Buffer for Y Low-Pass. */
104         static double accSumZ = 0; /* Buffer for Z Low-Pass. */
105         static double accFilterX = 0; /* Buffer for X Low-Pass. */
106         static double accFilterY = 0; /* Buffer for Y Low-Pass. */
107         static double accFilterZ = 0; /* Buffer for Z Low-Pass. */
108 
109         if (v == NULL) {
110             return ARGUMENT_ERROR;
111         }
112         if (twiStart(ACC_ADDRESS | TWI_WRITE)) {
113             return TWI_NO_ANSWER;
114         }
115         if (twiWrite(ACCREG_XL | (1 << 7))) { // Auto Increment
116             return TWI_WRITE_ERROR;
117         }
118         if (twiRepStart(ACC_ADDRESS | TWI_READ)) {
119             return TWI_NO_ANSWER;
120         }
121 
122         uint8_t xl = twiReadAck();
123         uint8_t xh = twiReadAck();
124         uint8_t yl = twiReadAck();
125         uint8_t yh = twiReadAck();
126         uint8_t zl = twiReadAck();
127         uint8_t zh = twiReadNak();
128 
129         int16_t x = *(int8_t *)(&xh);
130         x *= (1 << 8);
131         x |= xl;
132 
133         int16_t y = *(int8_t *)(&yh);
134         y *= (1 << 8);
135         y |= yl;
136 
137         int16_t z = *(int8_t *)(&zh);
138         z *= (1 << 8);
139         z |= zl;
140 
141         switch (accRange) {
142             case r2G:
143                 v->x = (((double)x) * 2 / 0x8000);
144                 v->y = (((double)y) * 2 / 0x8000);
145                 v->z = (((double)z) * 2 / 0x8000);
146                 break;
147             case r4G:
148                 v->x = (((double)x) * 4 / 0x8000);
149                 v->y = (((double)y) * 4 / 0x8000);
150                 v->z = (((double)z) * 4 / 0x8000);
151                 break;
152             case r8G:
153                 v->x = (((double)x) * 8 / 0x8000);
154                 v->y = (((double)y) * 8 / 0x8000);
155                 v->z = (((double)z) * 8 / 0x8000);
156                 break;
157             case r16G:
158                 v->x = (((double)x) * 16 / 0x8000);
159                 v->y = (((double)y) * 16 / 0x8000);
160                 v->z = (((double)z) * 16 / 0x8000);
161                 break;
162             default:
163                 return ARGUMENT_ERROR;
164         }
165 
166         accSumX = accSumX - accFilterX + v->x;
167         accFilterX = accSumX / ACCFILTERFACTOR;
168         v->x = accFilterX;
169 
170         accSumY = accSumY - accFilterY + v->y;
171         accFilterY = accSumY / ACCFILTERFACTOR;
172         v->y = accFilterY;
173 
174         accSumZ = accSumZ - accFilterZ + v->z;
```

```
    accFilterZ = accSumZ / ACCFILTERFACTOR;
    v->z = accFilterZ;

    return SUCCESS;
}
```

5.4.4.3 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 60 of file acc.c.

References TWI_NO_ANSWER.

Referenced by accInit().

```
60                                                    {
61     if (twiStart(ACC_ADDRESS | TWI_WRITE)) {
62         return TWI_NO_ANSWER;
63     }
64     if (twiWrite(reg)) {
65         return TWI_WRITE_ERROR;
66     }
67     if (twiWrite(val)) {
68         return TWI_WRITE_ERROR;
69     }
70     twiStop();
71     return SUCCESS;
72 }
```

5.4.5 Variable Documentation

5.4.5.1 AccRange accRange

Stored range to scale returned values.

Definition at line 51 of file acc.c.

Referenced by accInit(), and accRead().

5.5 Datatypes

Files

- file datatypes.h

Data Structures

- struct Vector3f

  The global 3-Dimensional Floating Point Vector.

5.5.1 Detailed Description

5.6 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.

5.6.1 Detailed Description

Error reporting with human readable strings.

5.6.2 Macro Definition Documentation

5.6.2.1 #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.

5.6.2.2 #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()

Definition at line 59 of file error.h.

5.6.3 Enumeration Type Documentation

5.6.3.1 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;
```

5.6.4 Function Documentation

5.6.4.1 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  }
```

5.7 Gyroscope Driver

Configuring and reading an L3GD20.

Files

- file gyro.h

  L3GD20 Gyroscope API Header.
- file gyro.c

  L3GD20 Gyroscope API Implementation.

Macros

- #define GYRO_ADDRESS 0xD6

  Gyroscope Address (110101xr, x = 1)
- #define GYROFILTERFACTOR 1

  Gyroscope Low Pass Factor.
- #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.

5.7.1 Detailed Description

Configuring and reading an L3GD20.

5.7.2 Macro Definition Documentation

5.7.2.1 #define GYRO_ADDRESS 0xD6

Gyroscope Address (110101xr, x = 1)

Definition at line 46 of file gyro.h.

Referenced by gyroRead().

5.7.2.2 #define GYROFILTERFACTOR 1

Gyroscope Low Pass Factor.

Definition at line 49 of file gyro.h.

Referenced by gyroRead().

5.7.2.3 #define GYROREG_CTRL1 0x20

Gyroscope Control Register 1.

Definition at line 47 of file gyro.c.

Referenced by gyroInit().

5.7.2.4 #define GYROREG_CTRL4 0x23

Gyroscope Control Register 4.

Definition at line 48 of file gyro.c.

Referenced by gyroInit().

5.7.2.5 #define GYROREG_OUTXL 0x28

First Gyroscope Output Register.

Definition at line 49 of file gyro.c.

Referenced by gyroRead().

5.7.3 Enumeration Type Documentation

5.7.3.1 enum GyroRange

Gyroscope Range options.

Enumerator

r250DPS	+- 250 Degrees per Second
r500DPS	+- 500 Degrees per Second
r2000DPS	+- 2000 Degrees per Second

Definition at line 52 of file gyro.h.

```
52               {
53     r250DPS,
54     r500DPS,
55     r2000DPS,
56 } GyroRange;
```

5.7.4 Function Documentation

5.7.4.1 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

Examples:

flight.c.

Definition at line 74 of file gyro.c.

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

```
74                              {
75     uint8_t v;
76     switch (r) {
77         case r250DPS:
78             v = 0x00;
79             break;
80         case r500DPS:
81             v = 0x10;
82             break;
83         case r2000DPS:
84             v = 0x20;
85             break;
86         default:
87             return ARGUMENT_ERROR;
88     }
89     gyroRange = r;
90     Error e = gyroWriteByte(GYROREG_CTRL1, 0x0F);
91     if (e != SUCCESS) {
92         return e;
93     }
94     e = gyroWriteByte(GYROREG_CTRL4, v);
95     return e;
96 }
```

5.7.4.2 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 98 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.

```
                           {
    // Simple Software Low-Pass
    static double gyroSumX = 0, gyroSumY = 0, gyroSumZ = 0;
    static double gyroFilterX = 0, gyroFilterY = 0, gyroFilterZ = 0;

    if (v == NULL) {
        return ARGUMENT_ERROR;
    }
    if (twiStart(GYRO_ADDRESS | TWI_WRITE)) {
        return TWI_NO_ANSWER;
    }
    if (twiWrite(GYROREG_OUTXL | 0x80)) { // Auto Increment
        return TWI_WRITE_ERROR;
    }
    if (twiRepStart(GYRO_ADDRESS | TWI_READ)) {
        return TWI_NO_ANSWER;
    }

    uint8_t xl = twiReadAck();
    uint8_t xh = twiReadAck();
    uint8_t yl = twiReadAck();
    uint8_t yh = twiReadAck();
    uint8_t zl = twiReadAck();
    uint8_t zh = twiReadNak();

    int16_t x = *(int8_t *)(&xh);
    x *= (1 << 8);
    x |= xl;

    int16_t y = *(int8_t *)(&yh);
    y *= (1 << 8);
    y |= yl;

    int16_t z = *(int8_t *)(&zh);
    z *= (1 << 8);
    z |= zl;

    switch (gyroRange) {
        case r250DPS:
            v->x = (((double)x) * 250 / 0x8000);
            v->y = (((double)y) * 250 / 0x8000);
            v->z = (((double)z) * 250 / 0x8000);
            break;
        case r500DPS:
            v->x = (((double)x) * 500 / 0x8000);
            v->y = (((double)y) * 500 / 0x8000);
            v->z = (((double)z) * 500 / 0x8000);
            break;
        case r2000DPS:
            v->x = (((double)x) * 2000 / 0x8000);
            v->y = (((double)y) * 2000 / 0x8000);
            v->z = (((double)z) * 2000 / 0x8000);
            break;
        default:
            return ARGUMENT_ERROR;
    }

    gyroSumX = gyroSumX - gyroFilterX + v->x;
    gyroFilterX = gyroSumX / GYROFILTERFACTOR;
    v->x = gyroFilterX;

    gyroSumY = gyroSumY - gyroFilterY + v->y;
    gyroFilterY = gyroSumY / GYROFILTERFACTOR;
    v->y = gyroFilterY;

    gyroSumZ = gyroSumZ - gyroFilterZ + v->z;
    gyroFilterZ = gyroSumZ / GYROFILTERFACTOR;
    v->z = gyroFilterZ;

    return SUCCESS;
}
```

5.7.4.3 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 60 of file gyro.c.

References TWI_NO_ANSWER.

Referenced by gyroInit().

```
60                                                         {
61      if (twiStart(GYRO_ADDRESS | TWI_WRITE)) {
62          return TWI_NO_ANSWER;
63      }
64      if (twiWrite(reg)) {
65          return TWI_WRITE_ERROR;
66      }
67      if (twiWrite(val)) {
68          return TWI_WRITE_ERROR;
69      }
70      twiStop();
71      return SUCCESS;
72 }
```

5.7.5 Variable Documentation

5.7.5.1 GyroRange gyroRange

Stored range to scale returned values.

Definition at line 51 of file gyro.c.

Referenced by gyroInit(), and gyroRead().

5.8 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.

5.8.1 Detailed Description

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

5.8.2 Enumeration Type Documentation

5.8.2.1 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;
```

5.8.3 Function Documentation

5.8.3.1 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 }
```

5.8.3.2 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().

```
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 }
```

5.8.3.3 void adcInit (ADCRef *ref*)

Initialize the ADC Hardware.

Parameters

ref	Reference Voltage.

Examples:

flight.c.

Definition at line 44 of file adc.c.

References AINT1, AINT2, AREF, and AVCC.

```
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 }
```

5.8.3.4 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().

```
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 }
```

5.8.3.5 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.

```
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 }
```

5.9 UART Library

UART Library enabling you to control all available UART Modules.

Files

- file serial.h

  UART Library Header File.
- file serial_device.h

  UART Library device-specific configuration.
- file serial.c

  UART Library Implementation.

Macros

- #define USB 0

  First UART Name.
- #define DISPLAY 1

  Second UART Name.
- #define RX_BUFFER_SIZE 128

  UART Receive Buffer Size.
- #define TX_BUFFER_SIZE 128

  UART Transmit Buffer Size.
- #define BAUD(baudRate, xtalCpu) ((xtalCpu)/((baudRate)∗16l)-1)

  Calculate Baudrate Register Value.
- #define RX_BUFFER_SIZE 32

  If you define this, a '\r' (CR) will be put in front of a '\n' (LF) when sending a byte.
- #define TX_BUFFER_SIZE 16

  TX Buffer Size in Bytes (Power of 2)
- #define FLOWCONTROL

  Defining this enables incoming XON XOFF (sends XOFF if rx buff is full)
- #define FLOWMARK 5

  Space remaining to trigger xoff/xon.
- #define XON 0x11

  XON Value.
- #define XOFF 0x13

  XOFF Value.

Functions

- uint8_t serialAvailable (void)

  Get number of available UART modules.
- uint8_t serialInit (uint8_t uart, uint16_t baud)

  Initialize the UART Hardware.
- void serialClose (uint8_t uart)

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

  Manually change the flow control.
- uint8_t serialHasChar (uint8_t uart)

  Check if a byte was received.

- uint8_t serialGet (uint8_t uart)

  Read a single byte.
- uint8_t serialGetBlocking (uint8_t uart)

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

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

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

  Send a byte.
- void serialWriteString (uint8_t uart, const char ∗data)

  Send a string.
- uint8_t serialTxBufferFull (uint8_t uart)

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

  Check if the transmit buffer is empty.

5.9.1 Detailed Description

UART Library enabling you to control all available UART Modules. With XON/XOFF Flow Control and buffered Receiving and Transmitting.

5.9.2 Macro Definition Documentation

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

Calculate Baudrate Register Value.

Examples:

flight.c.

Definition at line 52 of file serial.h.

5.9.2.2 #define DISPLAY 1

Second UART Name.

Definition at line 46 of file serial.h.

5.9.2.3 #define FLOWCONTROL

Defining this enables incoming XON XOFF (sends XOFF if rx buff is full)

Definition at line 64 of file serial.c.

5.9.2.4 #define FLOWMARK 5

Space remaining to trigger xoff/xon.

Definition at line 66 of file serial.c.

Referenced by serialGet().

5.9.2.5 #define RX_BUFFER_SIZE 128

UART Receive Buffer Size.

Definition at line 48 of file serial.h.

5.9.2.6 #define RX_BUFFER_SIZE 32

If you define this, a '\r' (CR) will be put in front of a '\n' (LF) when sending a byte.

Binary Communication will then be impossible!RX Buffer Size in Bytes (Power of 2)

Definition at line 56 of file serial.c.

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

5.9.2.7 #define TX_BUFFER_SIZE 128

UART Transmit Buffer Size.

Definition at line 49 of file serial.h.

5.9.2.8 #define TX_BUFFER_SIZE 16

TX Buffer Size in Bytes (Power of 2)

Definition at line 60 of file serial.c.

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

5.9.2.9 #define USB 0

First UART Name.

Definition at line 45 of file serial.h.

5.9.2.10 #define XOFF 0x13

XOFF Value.

Definition at line 68 of file serial.c.

Referenced by setFlow().

5.9.2.11 #define XON 0x11

XON Value.

Definition at line 67 of file serial.c.

Referenced by serialGet(), and setFlow().

5.9.3 Function Documentation

5.9.3.1 uint8_t serialAvailable (void)

Get number of available UART modules.

Returns

number of modules

Definition at line 115 of file serial.c.

```
115                             {
116     return UART_COUNT;
117 }
```

5.9.3.2 void serialClose (uint8_t *uart*)

Stop the UART Hardware.

Parameters

uart	UART Module to stop

Definition at line 172 of file serial.c.

References BANK_SERIAL, MEMSWITCH, MEMSWITCHBACK, and serialTxBufferEmpty().

```
172                              {
173     if (uart >= UART_COUNT)
174         return;
175 
176     MEMSWITCH(BANK_SERIAL);
177     uint8_t sreg = SREG;
178     sei();
179     while (!serialTxBufferEmpty(uart));
180     while (*serialRegisters[uart][SERIALB] & (1 << serialBits[uart][SERIALUDRIE])); // Wait while Transmit
    Interrupt is on
181     cli();
182     *serialRegisters[uart][SERIALB] = 0;
183     *serialRegisters[uart][SERIALC] = 0;
184     SREG = sreg;
185     free(rxBuffer[uart]);
186     free(txBuffer[uart]);
187     MEMSWITCHBACK(BANK_SERIAL);
188 }
```

5.9.3.3 uint8_t serialGet (uint8_t *uart*)

Read a single byte.

Parameters

uart	UART Module to read from

Returns

Received byte or 0

Definition at line 245 of file serial.c.

References BANK_SERIAL, FLOWMARK, MEMSWITCH, MEMSWITCHBACK, RX_BUFFER_SIZE, and XON.

Referenced by serialGetBlocking().

```
245                                {
246     if (uart >= UART_COUNT)
247         return 0;
248 
249     uint8_t c;
250     MEMSWITCH(BANK_SERIAL);
251 #ifdef FLOWCONTROL
252     rxBufferElements[uart]--;
253     if ((flow[uart] == 0) && (rxBufferElements[uart] <= FLOWMARK)) {
254         while (sendThisNext[uart] != 0);
255         sendThisNext[uart] = XON;
256         flow[uart] = 1;
257         if (shouldStartTransmission[uart]) {
258             shouldStartTransmission[uart] = 0;
259             *serialRegisters[uart][SERIALB] |= (1 << serialBits[uart][SERIALUDRIE]); // Enable Interrupt
260             *serialRegisters[uart][SERIALA] |= (1 << serialBits[uart][SERIALUDRE]); // Trigger Interrupt
261         }
262     }
263 #endif
264 
265     if (rxRead[uart] != rxWrite[uart]) {
266         c = rxBuffer[uart][rxRead[uart]];
267         rxBuffer[uart][rxRead[uart]] = 0;
268         if (rxRead[uart] < (RX_BUFFER_SIZE - 1)) {
269             rxRead[uart]++;
270         } else {
271             rxRead[uart] = 0;
272         }
273         MEMSWITCHBACK(BANK_SERIAL);
274         return c;
275     } else {
276         MEMSWITCHBACK(BANK_SERIAL);
277         return 0;
278     }
279 }
```

5.9.3.4 uint8_t serialGetBlocking (uint8_t *uart*)

Wait until a character is received.

Parameters

uart	UART Module to read from

Returns

Received byte

Definition at line 237 of file serial.c.

References serialGet(), and serialHasChar().

```
237                                         {
238     if (uart >= UART_COUNT)
239         return 0;
240 
241     while(!serialHasChar(uart));
242     return serialGet(uart);
243 }
```

5.9.3.5 uint8_t serialHasChar (uint8_t *uart*)

Check if a byte was received.

Parameters

uart	UART Module to check

Returns

1 if a byte was received, 0 if not

Definition at line 226 of file serial.c.

Referenced by serialGetBlocking().

```
226                                    {
227     if (uart >= UART_COUNT)
228         return 0;
229
230     if (rxRead[uart] != rxWrite[uart]) { // True if char available
231         return 1;
232     } else {
233         return 0;
234     }
235 }
```

5.9.3.6 uint8_t serialInit (uint8_t *uart,* uint16_t *baud*)

Initialize the UART Hardware.

Parameters

uart	UART Module to initialize
baud	Baudrate. Use the BAUD() macro!

Returns

1 if not enough memory for buffers, 0 on success

Examples:

flight.c.

Definition at line 119 of file serial.c.

References BANK_SERIAL, MEMSWITCH, MEMSWITCHBACK, RX_BUFFER_SIZE, and TX_BUFFER_SIZE.

```
119                                                 {
120     if (uart >= UART_COUNT) {
121         for (uint8_t i = 0; i < UART_COUNT; i++) {
122             serialInit(i, baud);
123         }
124     }
125
126     MEMSWITCH(BANK_SERIAL);
127
128     rxBuffer[uart] = (uint8_t *)malloc(RX_BUFFER_SIZE);
129     if (rxBuffer[uart] == NULL) {
130         MEMSWITCHBACK(BANK_SERIAL);
131         return 1;
132     }
133
134     txBuffer[uart] = (uint8_t *)malloc(TX_BUFFER_SIZE);
135     if (txBuffer[uart] == NULL) {
136         free((void *)rxBuffer[uart]);
137         MEMSWITCHBACK(BANK_SERIAL);
138         return 1;
139     }
140
141     // Initialize state variables
142     rxRead[uart] = 0;
143     rxWrite[uart] = 0;
144     txRead[uart] = 0;
145     txWrite[uart] = 0;
146     shouldStartTransmission[uart] = 1;
147 #ifdef FLOWCONTROL
148     sendThisNext[uart] = 0;
149     flow[uart] = 1;
150     rxBufferElements[uart] = 0;
151 #endif
152
153     // Default Configuration: 8N1
154     *serialRegisters[uart][SERIALC] = (1 << serialBits[uart][SERIALUCSZ0]) | (1 << serialBits[uart][
      SERIALUCSZ1]);
155
156     // Set baudrate
157 #if SERIALBAUDBIT == 8
158     *serialRegisters[uart][SERIALUBRRH] = (baud >> 8);
159     *serialRegisters[uart][SERIALUBRRL] = baud;
160 #else
161     *serialBaudRegisters[uart] = baud;
162 #endif
163
164     *serialRegisters[uart][SERIALB] = (1 << serialBits[uart][SERIALRXCIE]); // Enable Interrupts
165     *serialRegisters[uart][SERIALB] |= (1 << serialBits[uart][SERIALRXEN]) | (1 << serialBits[uart][SERIALTXEN]); // Enable Receiver/Transmitter
166
167     MEMSWITCHBACK(BANK_SERIAL);
168
169     return 0;
170 }
```

5.9.3.7 uint8_t serialRxBufferEmpty (uint8_t *uart*)

Check if the receive buffer is empty.

Parameters

uart	UART Module to check

Returns

1 if buffer is empty, 0 if not.

Definition at line 288 of file serial.c.

```
288                                           {
289     if (uart >= UART_COUNT)
290         return 0;
291
292     if (rxRead[uart] != rxWrite[uart]) {
293         return 0;
294     } else {
295         return 1;
296     }
297 }
```

5.9.3.8 uint8_t serialRxBufferFull (uint8_t *uart*)

Check if the receive buffer is full.

Parameters

uart	UART Module to check

Returns

1 if buffer is full, 0 if not

Definition at line 281 of file serial.c.

References RX_BUFFER_SIZE.

```
281                                          {
282     if (uart >= UART_COUNT)
283         return 0;
284
285     return (((rxWrite[uart] + 1) == rxRead[uart]) || ((rxRead[uart] == 0) && ((rxWrite[uart] + 1) == RX_BUFFER_SIZE)));
286 }
```

5.9.3.9 uint8_t serialTxBufferEmpty (uint8_t *uart*)

Check if the transmit buffer is empty.

Parameters

uart	UART Module to check

Returns

1 if buffer is empty, 0 if not.

Definition at line 349 of file serial.c.

Referenced by serialClose().

```
349                                         {
350     if (uart >= UART_COUNT)
351         return 0;
352
353     if (txRead[uart] != txWrite[uart]) {
354         return 0;
355     } else {
356         return 1;
357     }
358 }
```

5.9.3.10 uint8_t serialTxBufferFull (uint8_t *uart*)

Check if the transmit buffer is full.

Parameters

uart	UART Module to check

Returns

1 if buffer is full, 0 if not

Definition at line 342 of file serial.c.

References TX_BUFFER_SIZE.

Referenced by serialWrite().

```
342                                        {
343     if (uart >= UART_COUNT)
344         return 0;
345
346     return (((txWrite[uart] + 1) == txRead[uart]) || ((txRead[uart] == 0) && ((txWrite[uart] + 1) ==
      TX_BUFFER_SIZE)));
347 }
```

5.9.3.11 void serialWrite (uint8_t *uart,* uint8_t *data*)

Send a byte.

Parameters

uart	UART Module to write to
data	Byte to send

Definition at line 303 of file serial.c.

References BANK_SERIAL, MEMSWITCH, MEMSWITCHBACK, serialTxBufferFull(), and TX_BUFFER_SIZE.

Referenced by serialWriteString().

```
303                                             {
304     if (uart >= UART_COUNT)
305         return;
306 
307     MEMSWITCH(BANK_SERIAL);
308 #ifdef SERIALINJECTCR
309     if (data == ’\n’) {
310         serialWrite(uart, ’\r’);
311     }
312 #endif
313     while (serialTxBufferFull(uart));
314 
315     txBuffer[uart][txWrite[uart]] = data;
316     if (txWrite[uart] < (TX_BUFFER_SIZE - 1)) {
317         txWrite[uart]++;
318     } else {
319         txWrite[uart] = 0;
320     }
321     if (shouldStartTransmission[uart]) {
322         shouldStartTransmission[uart] = 0;
323         *serialRegisters[uart][SERIALB] |= (1 << serialBits[uart][SERIALUDRIE]); // Enable Interrupt
324         *serialRegisters[uart][SERIALA] |= (1 << serialBits[uart][SERIALUDRE]); // Trigger Interrupt
325     }
326     MEMSWITCHBACK(BANK_SERIAL);
327 }
```

5.9.3.12 void serialWriteString (uint8_t *uart,* const char * *data*)

Send a string.

Parameters

uart	UART Module to write to
data	Null-Terminated String

Definition at line 329 of file serial.c.

References serialWrite().

```
329                                                       {
330     if (uart >= UART_COUNT)
331         return;
332 
333     if (data == 0) {
334         serialWriteString(uart, "NULL");
335     } else {
336         while (*data != ’\0’) {
337             serialWrite(uart, *data++);
338         }
339     }
340 }
```

5.9.3.13 void setFlow (uint8_t *uart,* uint8_t *on*)

Manually change the flow control.

Flow Control has to be compiled into the library!

Parameters

uart	UART Module to operate on
on	1 of on, 0 if off

Definition at line 191 of file serial.c.

References XOFF, and XON.

```
191                                       {
192     if (uart >= UART_COUNT)
```

```
        return;

    if (flow[uart] != on) {
        if (on == 1) {
            // Send XON
            while (sendThisNext[uart] != 0);
            sendThisNext[uart] = XON;
            flow[uart] = 1;
            if (shouldStartTransmission[uart]) {
                shouldStartTransmission[uart] = 0;
                *serialRegisters[uart][SERIALB] |= (1 << serialBits[uart][SERIALUDRIE]);
                *serialRegisters[uart][SERIALA] |= (1 << serialBits[uart][SERIALUDRE]); // Trigger Interrupt
            }
        } else {
            // Send XOFF
            sendThisNext[uart] = XOFF;
            flow[uart] = 0;
            if (shouldStartTransmission[uart]) {
                shouldStartTransmission[uart] = 0;
                *serialRegisters[uart][SERIALB] |= (1 << serialBits[uart][SERIALUDRIE]);
                *serialRegisters[uart][SERIALA] |= (1 << serialBits[uart][SERIALUDRE]); // Trigger Interrupt
            }
        }
        // Wait till it's transmitted
        while (*serialRegisters[uart][SERIALB] & (1 << serialBits[uart][SERIALUDRIE]));
    }
}
```

5.10 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.

5.10.1 Detailed Description

Measuring Time with Millisecond Resolution. Uses Timer 2

Prescaler 64

Count to 250

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

5.10.2 Macro Definition Documentation

5.10.2.1 #define OCIE OCIE2A

Timer 2 Compare Match A Interrupt Enable.

Definition at line 53 of file time.c.

5.10.2.2 #define OCR OCR2A

Timer 2 Compare Register A.

Definition at line 51 of file time.c.

5.10.2.3 #define TCRA TCCR2A

Timer 2 Control Register A.

Definition at line 49 of file time.c.

5.10.2.4 #define TCRB TCCR2B

Timer 2 Control Register B.

Definition at line 50 of file time.c.

5.10.2.5 #define TIMS TIMSK2

Timer 2 Interrupt Mask.

Definition at line 52 of file time.c.

5.10.3 Typedef Documentation

5.10.3.1 typedef uint64_t time_t

Timekeeping Data Type.

Overflows after 500 million years... :)

Definition at line 53 of file time.h.

5.10.4 Function Documentation

5.10.4.1 time_t getSystemTime (void)

Get the System Uptime.

Returns

System Uptime in Milliseconds

Definition at line 68 of file time.c.

References systemTime.

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

5.10.4.2 void initSystemTimer (void)

Initialize the system timer.

Execution every millisecond. Uses Timer 2.

Examples:

flight.c.

Definition at line 55 of file time.c.

```
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 }
```

5.10.4.3 ISR (TIMER2_COMPA_vect)

Timer 2 Compare Match A Interrupt.

Definition at line 64 of file time.c.

References systemTime.

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

5.10.5 Variable Documentation

5.10.5.1 volatile time_t systemTime = 0

Current System Uptime.

Definition at line 47 of file time.c.

Referenced by getSystemTime(), and ISR().

5.11 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.

5.11.1 Detailed Description

Using the AVR TWI/I2C Hardware.

5.11.2 Macro Definition Documentation

5.11.2.1 #define TWI_READ 1

I2C Read Bit.

Definition at line 43 of file twi.h.

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

5.11.2.2 #define TWI_WRITE 0

I2C Write Bit.

Definition at line 44 of file twi.h.

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

5.11.3 Function Documentation

5.11.3.1 void twiInit (void)

Initialize the I2C Hardware.

Examples:

flight.c.

Definition at line 26 of file twi.c.

```
{
  /* initialize TWI clock: 100 kHz clock, TWPS = 0 => prescaler = 1 */

  TWSR = 0;                         /* no prescaler */
  TWBR = ((F_CPU/SCL_CLOCK)-16)/2;  /* must be > 10 for stable operation */

}/* i2c_init */
```

5.11.3.2 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().

```
{
    TWCR = (1<<TWINT) | (1<<TWEN) | (1<<TWEA);
    while(!(TWCR & (1<<TWINT)));

    return TWDR;

}/* i2c_readAck */
```

5.11.3.3 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().

```
{
    TWCR = (1<<TWINT) | (1<<TWEN);
    while(!(TWCR & (1<<TWINT)));

    return TWDR;

}/* i2c_readNak */
```

5.11.3.4 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().

```
{
    return twiStart( address );

}/* i2c_rep_start */
```

5.11.3.5 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(), magWriteRegister(), and twiRepStart().

```
{
    uint8_t   twst;

    // send START condition
    TWCR = (1<<TWINT) | (1<<TWSTA) | (1<<TWEN);

    // wait until transmission completed
    while(!(TWCR & (1<<TWINT)));

    // check value of TWI Status Register. Mask prescaler bits.
    twst = TW_STATUS & 0xF8;
    if ( (twst != TW_START) && (twst != TW_REP_START)) return 1;

    // send device address
    TWDR = address;
    TWCR = (1<<TWINT) | (1<<TWEN);

    // wail until transmission completed and ACK/NACK has been received
    while(!(TWCR & (1<<TWINT)));

    // check value of TWI Status Register. Mask prescaler bits.
    twst = TW_STATUS & 0xF8;
    if ( (twst != TW_MT_SLA_ACK) && (twst != TW_MR_SLA_ACK) ) return 1;

    return 0;

}/* i2c_start */
```

5.11.3.6 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 */
```

5.11.3.7 void twiStop (void)

Stop the I2C Hardware.

Definition at line 137 of file twi.c.

Referenced by magWriteRegister().

```
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 */
```

5.11.3.8 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 magWriteRegister().

```
{
    uint8_t   twst;

    // send data to the previously addressed device
    TWDR = data;
    TWCR = (1<<TWINT) | (1<<TWEN);

    // wait until transmission completed
    while(!(TWCR & (1<<TWINT)));

    // check value of TWI Status Register. Mask prescaler bits
    twst = TW_STATUS & 0xF8;
    if( twst != TW_MT_DATA_ACK) return 1;
    return 0;

}/* i2c_write */
```

5.12 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.
- #define BANK_SERIAL 0

  Bank for serial buffers.
- #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.

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)

5.12.1 Detailed Description

Allows access to external RAM with bank-switching.

5.12.2 Macro Definition Documentation

5.12.2.1 #define BANK0DDR DDRG

First Bank Selection Data Direction Register.

Definition at line 59 of file xmem.h.

Referenced by xmemInit().

5.12.2.2 #define BANK0PIN PG3

First Bank Selection Pin.

Definition at line 60 of file xmem.h.

Referenced by xmemInit(), and xmemSetBank().

5.12.2.3 #define BANK0PORT PORTG

First Bank Selection Port.

Definition at line 58 of file xmem.h.

Referenced by xmemInit(), and xmemSetBank().

5.12.2.4 #define BANK1DDR DDRG

Second Bank Selection Data Direction Register.

Definition at line 62 of file xmem.h.

Referenced by xmemInit().

5.12.2.5 #define BANK1PIN PG4

Second Bank Selection Pin.

Definition at line 63 of file xmem.h.

Referenced by xmemInit(), and xmemSetBank().

5.12.2.6 #define BANK1PORT PORTG

Second Bank Selection Port.

Definition at line 61 of file xmem.h.

Referenced by xmemInit(), and xmemSetBank().

5.12.2.7 #define BANK2DDR DDRL

Third Bank Selection Data Direction Register.

Definition at line 65 of file xmem.h.

Referenced by xmemInit().

5.12.2.8 #define BANK2PIN PL5

Third Bank Selection Pin.

Definition at line 66 of file xmem.h.

Referenced by xmemInit(), and xmemSetBank().

5.12.2.9 #define BANK2PORT PORTL

Third Bank Selection Port.

Definition at line 64 of file xmem.h.

Referenced by xmemInit(), and xmemSetBank().

5.12.2.10 #define BANK_GENERIC 0

Generic Memory Bank.

Definition at line 55 of file xmem.h.

5.12.2.11 #define BANK_SERIAL 0

Bank for serial buffers.

Definition at line 56 of file xmem.h.

Referenced by serialClose(), serialGet(), serialInit(), and serialWrite().

5.12.2.12 #define MEMBANKS 8

Available Memory Banks.

Definition at line 54 of file xmem.h.

Referenced by xmemInit(), and xmemSetBank().

5.12.2.13 #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 serialClose(), serialGet(), serialInit(), and serialWrite().

5.12.2.14 #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 serialClose(), serialGet(), serialInit(), and serialWrite().

5.12.3 Function Documentation

5.12.3.1 void restoreState (uint8_t *bank*)

Restore the malloc state.

Parameters

bank	Location of state to load.

Definition at line 64 of file xmem.c.

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

Referenced by xmemSetBank().

```
64                               {
65      __malloc_heap_start = states[bank].start;
66      __malloc_heap_end = states[bank].end;
67      __brkval = states[bank].val;
68     __flp = states[bank].fl;
69 }
```

5.12.3.2 void saveState (uint8_t *bank*)

Save the current malloc state.

Parameters

bank	Current Bank Number

Definition at line 54 of file xmem.c.

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

Referenced by xmemInit(), and xmemSetBank().

```
54                              {
55     states[bank].start = __malloc_heap_start;
56     states[bank].end = __malloc_heap_end;
57     states[bank].val = __brkval;
58     states[bank].fl = __flp;
59 }
```

5.12.3.3 uint8_t xmemGetBank (void)

Get the current memory bank.

Returns

Current Memory Bank.

Definition at line 104 of file xmem.c.

References currentBank.

```
104                         {
105     return currentBank;
106 }
```

5.12.3.4 void xmemInit (void)

Initialize the External Memory Interface.

Examples:

flight.c.

Definition at line 71 of file xmem.c.

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

```
71                      {
72     BANK0DDR |= (1 << BANK0PIN);
73     BANK1DDR |= (1 << BANK1PIN);
74     BANK2DDR |= (1 << BANK2PIN);
75     BANK0PORT &= ~(1 << BANK0PIN);
76     BANK1PORT &= ~(1 << BANK1PIN);
77     BANK2PORT &= ~(1 << BANK2PIN);
78 
79     XMCRB = 0; // Use full address space
80     XMCRA = (1 << SRW11) | (1 << SRW10); // 3 Wait cycles
81     XMCRA |= (1 << SRE); // Enable XMEM
82
83     for (uint8_t i = 0; i < MEMBANKS; i++) {
84         saveState(i);
85     }
86 }
```

5.12.3.5 void xmemSetBank (uint8_t *bank*)

Switch the active memory bank.

Parameters

bank	New Memory Bank

Definition at line 88 of file xmem.c.

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

```
88                                  {
89     if (bank < MEMBANKS) {
90         saveState(currentBank);
91
92         BANK0PORT &= ~(1 << BANK0PIN);
93         BANK1PORT &= ~(1 << BANK1PIN);
94         BANK2PORT &= ~(1 << BANK2PIN);
95         BANK0PORT |= ((bank & 0x01) << BANK0PIN);
96         BANK1PORT |= (((bank & 0x02) >> 1) << BANK1PIN);
97         BANK2PORT |= (((bank & 0x04) >> 2) << BANK2PIN);
98
99         currentBank = bank;
100         restoreState(bank);
101     }
102 }
```

5.12.4 Variable Documentation

5.12.4.1 void∗ __brkval

Internal Malloc Heap-End Pointer.

Referenced by restoreState(), and saveState().

5.12.4.2 void∗ __flp

Internal Malloc Free List Pointer (State)

Referenced by restoreState(), and saveState().

5.12.4.3 uint8_t currentBank = 0

Current active Memory Bank.

Definition at line 46 of file xmem.c.

Referenced by xmemGetBank(), and xmemSetBank().

5.12.4.4 uint8_t currentBank

Current active Memory Bank.

Definition at line 46 of file xmem.c.

Referenced by xmemGetBank(), and xmemSetBank().

5.12.4.5 MallocState states[MEMBANKS]

MallocState for all Memory Banks.

Definition at line 45 of file xmem.c.

5.12.4.6 MallocState states[MEMBANKS]

MallocState for all Memory Banks.

Definition at line 45 of file xmem.c.

5.13 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 MAG_ADDRESS 0x3C

  Magnetometer Address.
- #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.

Variables

- MagRange magRange

  Stored range to scale returned values.

5.13.1 Detailed Description

Configuring and reading an LSM303DLHC Magnetometer.

5.13.2 Macro Definition Documentation

5.13.2.1 #define MAG_ADDRESS 0x3C

Magnetometer Address.

Definition at line 46 of file mag.h.

Referenced by magRead(), and magWriteRegister().

5.13.2.2 #define MAGREG_CRB 0x01

Magnetometer Gain Register.

Definition at line 47 of file mag.c.

Referenced by magInit().

5.13.2.3 #define MAGREG_MR 0x02

Magnetometer Mode Register.

Definition at line 48 of file mag.c.

Referenced by magInit().

5.13.2.4 #define MAGREG_XH 0x03

First Magnetometer Output Register.

Definition at line 49 of file mag.c.

Referenced by magRead().

5.13.3 Enumeration Type Documentation

5.13.3.1 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 49 of file mag.h.

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

5.13.4 Function Documentation

5.13.4.1 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.

Examples:

flight.c.

Definition at line 76 of file mag.c.

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

```
76                              {
77     if ((r <= 0) || (r >= 8)) {
78         return ARGUMENT_ERROR;
79     }
80     Error e = magWriteRegister(MAGREG_MR, 0x00); // Continuous Conversion
81     if (e != SUCCESS) {
82         return e;
83     }
84     e = magWriteRegister(MAGREG_CRB, (r << 5)); // Set Range
85     magRange = r;
86     return e;
87 }
```

5.13.4.2 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 89 of file mag.c.

References ARGUMENT_ERROR, MAG_ADDRESS, magRange, MAGREG_XH, r1g3, r1g9, r2g5, r4g0, r4g7, r5g6, r8g1, SUCCESS, TWI_NO_ANSWER, TWI_READ, TWI_WRITE, TWI_WRITE_ERROR, twiReadAck(), twiReadNak(), twiRepStart(), twiStart(), twiWrite(), Vector3f::x, Vector3f::y, and Vector3f::z.

```
89                               {
90     if (v == NULL) {
91         return ARGUMENT_ERROR;
92     }
93     if (twiStart(MAG_ADDRESS | TWI_WRITE)) {
94         return TWI_NO_ANSWER;
95     }
```

```
    if (twiWrite(MAGREG_XH)) {
        return TWI_WRITE_ERROR;
    }
    if (twiRepStart(MAG_ADDRESS | TWI_READ)) {
        return TWI_NO_ANSWER;
    }
    uint8_t xh = twiReadAck();
    uint8_t xl = twiReadAck();
    uint8_t zh = twiReadAck();
    uint8_t zl = twiReadAck();
    uint8_t yh = twiReadAck();
    uint8_t yl = twiReadNak();

    int16_t x = *(int8_t *)(&xh);
    x *= (1 << 8);
    x |= xl;

    int16_t y = *(int8_t *)(&yh);
    y *= (1 << 8);
    y |= yl;

    int16_t z = *(int8_t *)(&zh);
    z *= (1 << 8);
    z |= zl;

    switch (magRange) {
        case r1g3:
            v->x = (((double)x) * 1.3 / MAG_NORMALIZE);
            v->y = (((double)y) * 1.3 / MAG_NORMALIZE);
            v->z = (((double)z) * 1.3 / MAG_NORMALIZE);
            break;
        case r1g9:
            v->x = (((double)x) * 1.9 / MAG_NORMALIZE);
            v->y = (((double)y) * 1.9 / MAG_NORMALIZE);
            v->z = (((double)z) * 1.9 / MAG_NORMALIZE);
            break;
        case r2g5:
            v->x = (((double)x) * 2.5 / MAG_NORMALIZE);
            v->y = (((double)y) * 2.5 / MAG_NORMALIZE);
            v->z = (((double)z) * 2.5 / MAG_NORMALIZE);
            break;
        case r4g0:
            v->x = (((double)x) * 4.0 / MAG_NORMALIZE);
            v->y = (((double)y) * 4.0 / MAG_NORMALIZE);
            v->z = (((double)z) * 4.0 / MAG_NORMALIZE);
            break;
        case r4g7:
            v->x = (((double)x) * 4.7 / MAG_NORMALIZE);
            v->y = (((double)y) * 4.7 / MAG_NORMALIZE);
            v->z = (((double)z) * 4.7 / MAG_NORMALIZE);
            break;
        case r5g6:
            v->x = (((double)x) * 5.6 / MAG_NORMALIZE);
            v->y = (((double)y) * 5.6 / MAG_NORMALIZE);
            v->z = (((double)z) * 5.6 / MAG_NORMALIZE);
            break;
        case r8g1:
            v->x = (((double)x) * 8.1 / MAG_NORMALIZE);
            v->y = (((double)y) * 8.1 / MAG_NORMALIZE);
            v->z = (((double)z) * 8.1 / MAG_NORMALIZE);
            break;
        default:
            return ARGUMENT_ERROR;
    }

    return SUCCESS;
}
```

5.13.4.3 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 62 of file mag.c.

References MAG_ADDRESS, SUCCESS, TWI_NO_ANSWER, TWI_WRITE, TWI_WRITE_ERROR, twiStart(), twiStop(), and twiWrite().

Referenced by magInit().

```
62                                              {
63     if (twiStart(MAG_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 }
```

5.13.5 Variable Documentation

5.13.5.1 MagRange magRange

Stored range to scale returned values.

Definition at line 53 of file mag.c.

Referenced by magInit(), and magRead().

5.14 Remote Control Interface

Read RC Receiver Sum Signal.

Files

- file remote.h

  Remote API Header.
- file remote.c

  Remote API Implementation.

Macros

- #define RC_EXTINT 4

  External Interrupt connected to sum signal.
- #define RC_CHANNELS 6

  Number of Channels in RC Receiver.

Functions

- void rcInit (void)

  Initialize RC Receiver.
- ISR (TIMER0_OVF_vect)

  External Interrupt detecting Timer0 Overflow, invalidating signal matching.
- ISR (INTn_vect)

  External Interrupt Service Routine for received sum signal.

Variables

- volatile int8_t rcValues [RC_CHANNELS]

  Stick positions of remote control.
- volatile int8_t rcValues [RC_CHANNELS]

  Stick positions of remote control.

5.14.1 Detailed Description

Read RC Receiver Sum Signal.

5.14.2 Macro Definition Documentation

5.14.2.1 #define RC_CHANNELS 6

Number of Channels in RC Receiver.

Definition at line 45 of file remote.h.

5.14.2.2 #define RC_EXTINT 4

External Interrupt connected to sum signal.

Definition at line 43 of file remote.h.

5.14.3 Function Documentation

5.14.3.1 ISR (TIMER0_OVF_vect)

External Interrupt detecting Timer0 Overflow, invalidating signal matching.

Definition at line 62 of file remote.c.

```
62                        {
63     rcSignalCounter = 0; // Reset Channel Counter
64     rcSignalValid = 0; // Block measurement until next pulse starts
65 }
```

5.14.3.2 ISR (INTn_vect)

External Interrupt Service Routine for received sum signal.

Definition at line 68 of file remote.c.

References rcValues.

```
68                  {
69     if (rcSignalValid == 1) { // Start with first pulse
70         rcValues[rcSignalCounter] = TCNT0; // Store pulse length
71     } else {
72         rcSignalValid = 1; // First pulse received, start!
73     }
74     TCNT0 = RC_TIMER_RELOAD; // Reset Timer
75 }
```

5.14.3.3 void rcInit (void)

Initialize RC Receiver.

Examples:

flight.c.

Definition at line 77 of file remote.c.

```
77                   {
78     TIMSK0 |= (1 << TOIE0); // Enable Overflow Interrupt
79     TCNT0 = RC_TIMER_RELOAD; // Overflow after 3,68ms
80     TCCR0B |= (1 << CS02); // Prescaler 256
81 
82     // Select matching External Interrupt Control Register
83 #if RC_EXTINT < 4
84     volatile uint8_t *extIntReg = &EICRA;
85 #elif RC_EXTINT < 8
86     volatile uint8_t *extIntReg = &EICRB;
87 #else
88 #error EXTINT too high!
89 #endif
90 
91     *extIntReg |= (1 << ISCn0) | (1 << ISCn1); // Trigger on rising edge
92     EIMSK |= (1 << INTn); // Enable external interrupt
93 }
```

5.14.4 Variable Documentation

5.14.4.1 volatile int8_t rcValues[RC_CHANNELS]

Stick positions of remote control.

Definition at line 54 of file remote.c.

Referenced by ISR().

5.14.4.2 volatile int8_t rcValues[RC_CHANNELS]

Stick positions of remote control.

Definition at line 54 of file remote.c.

Referenced by ISR().

Chapter 6

Data Structure Documentation

6.1 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.

6.1.1 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 73 of file xmem.h.

6.1.2 Field Documentation

6.1.2.1 char∗ end

End of Heap.

Definition at line 75 of file xmem.h.

Referenced by restoreState(), and saveState().

6.1.2.2 void∗ fl

Free List.

Definition at line 77 of file xmem.h.

Referenced by restoreState(), and saveState().

6.1.2.3 char∗ start

Start of Heap.

Definition at line 74 of file xmem.h.

Referenced by restoreState(), and saveState().

6.1.2.4 void∗ val

Highest Heap Point.

Definition at line 76 of file xmem.h.

Referenced by restoreState(), and saveState().

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

- include/lowlevel/xmem.h

6.2 Vector3f Struct Reference

The global 3-Dimensional Floating Point Vector.

`#include <datatypes.h>`

Data Fields

- double x

  X Part.
- double y

  Y Part.
- double z

  Z Part.

6.2.1 Detailed Description

The global 3-Dimensional Floating Point Vector.

Definition at line 42 of file datatypes.h.

6.2.2 Field Documentation

6.2.2.1 double x

X Part.

Definition at line 43 of file datatypes.h.

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

6.2.2.2 double y

Y Part.

Definition at line 44 of file datatypes.h.

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

6.2.2.3 double z

Z Part.

Definition at line 45 of file datatypes.h.

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

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

- include/datatypes.h

Chapter 7

File Documentation

7.1 include/acc.h File Reference

LSM303DLHC Accelerometer API Header.

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

Macros

- #define ACC_ADDRESS 0x32

  Accelerometer Address (0011001r)
- #define ACCFILTERFACTOR 1

  Accelerometer Low Pass Factor.

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/datatypes.h File Reference

Data Structures

- struct Vector3f

  The global 3-Dimensional Floating Point Vector.

7.3 include/doc.h File Reference

Contains Doxygen Group Definitions.

7.3.1 Detailed Description

Contains Doxygen Group Definitions.

Definition in file doc.h.

7.4 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.

7.4.1 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.

7.5 include/gyro.h File Reference

L3GD20 Gyroscope API Header.

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

Macros

- #define GYRO_ADDRESS 0xD6

  Gyroscope Address (110101xr, x = 1)
- #define GYROFILTERFACTOR 1

  Gyroscope Low Pass Factor.

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.

7.5.1 Detailed Description

L3GD20 Gyroscope API Header.

Definition in file gyro.h.

7.6 include/lowlevel/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.

7.6.1 Detailed Description

Analog-to-Digital Converter API Header.

Definition in file adc.h.

7.7 include/lowlevel/serial.h File Reference

UART Library Header File.

Macros

- #define USB 0

  First UART Name.
- #define DISPLAY 1

  Second UART Name.
- #define RX_BUFFER_SIZE 128

  UART Receive Buffer Size.
- #define TX_BUFFER_SIZE 128

  UART Transmit Buffer Size.
- #define BAUD(baudRate, xtalCpu) ((xtalCpu)/((baudRate)∗16l)-1)

  Calculate Baudrate Register Value.

Functions

- uint8_t serialAvailable (void)

  Get number of available UART modules.
- uint8_t serialInit (uint8_t uart, uint16_t baud)

  Initialize the UART Hardware.
- void serialClose (uint8_t uart)

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

  Manually change the flow control.
- uint8_t serialHasChar (uint8_t uart)

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

  Read a single byte.
- uint8_t serialGetBlocking (uint8_t uart)

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

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

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

Send a byte.

- void serialWriteString (uint8_t uart, const char ∗data)

  Send a string.
- uint8_t serialTxBufferFull (uint8_t uart)

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

  Check if the transmit buffer is empty.

7.7.1 Detailed Description

UART Library Header File.

Definition in file serial.h.

7.8 include/lowlevel/serial_device.h File Reference

UART Library device-specific configuration.

7.8.1 Detailed Description

UART Library device-specific configuration. Contains Register and Bit Positions for different AVR devices.

Definition in file serial_device.h.

7.9 include/lowlevel/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.

7.9.1 Detailed Description

Time API Header.

Definition in file time.h.

7.10 include/lowlevel/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.

7.10.1 Detailed Description

I2C API Header.

Definition in file twi.h.

7.11 include/lowlevel/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.
- #define BANK_SERIAL 0

  Bank for serial buffers.
- #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.

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.

7.11.1 Detailed Description

XMEM API Header.

Definition in file xmem.h.

7.12 include/mag.h File Reference

LSM303DLHC Magnetometer API Header.

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

Macros

- #define MAG_ADDRESS 0x3C

  Magnetometer Address.

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.

7.12.1 Detailed Description

LSM303DLHC Magnetometer API Header.

Definition in file mag.h.

7.13 include/remote.h File Reference

Remote API Header.

Macros

- #define RC_EXTINT 4

  External Interrupt connected to sum signal.
- #define RC_CHANNELS 6

  Number of Channels in RC Receiver.

Functions

- void rcInit (void)

  Initialize RC Receiver.

Variables

- volatile int8_t rcValues [RC_CHANNELS]

  Stick positions of remote control.

7.13.1 Detailed Description

Remote API Header.

Definition in file remote.h.

7.14 lib/acc.c File Reference

LSM303DLHC Accelerometer API Implementation.

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

7.14.1 Detailed Description

LSM303DLHC Accelerometer API Implementation.

Definition in file acc.c.

7.15 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 [] = "SUCC"

  String for SUCCESS.
- char PROGMEM error1 [] = "TWI_ANSWER"

  String for TWI_NO_ANSWER.
- char PROGMEM error2 [] = "TWI_WRITE"

  String for TWI_WRITE_ERROR.
- char PROGMEM error3 [] = "NO_MEM"

  String for MALLOC_FAIL.
- char PROGMEM error4 [] = "ERROR"

  String for ERROR.
- char PROGMEM error5 [] = "ARG"

  String for ARGUMENT_ERROR.
- PGM_P PROGMEM errorTable []

  Array of all error descriptions in Flash Memory.

7.15.1 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.

7.15.2 Variable Documentation

7.15.2.1 char PROGMEM error0[] = ”SUCC”

String for SUCCESS.

Definition at line 43 of file error.c.

7.15.2.2 char PROGMEM error1[] = ”TWI_ANSWER”

String for TWI_NO_ANSWER.

Definition at line 44 of file error.c.

7.15.2.3 char PROGMEM error2[] = ”TWI_WRITE”

String for TWI_WRITE_ERROR.

Definition at line 45 of file error.c.

7.15.2.4 char PROGMEM error3[] = ”NO_MEM”

String for MALLOC_FAIL.

Definition at line 46 of file error.c.

7.15.2.5 char PROGMEM error4[ ] = "ERROR"

String for ERROR.

Definition at line 47 of file error.c.

7.15.2.6 char PROGMEM error5[ ] = "ARG"

String for ARGUMENT_ERROR.

Definition at line 48 of file error.c.

7.15.2.7 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().

7.16 lib/gyro.c File Reference

L3GD20 Gyroscope API Implementation.

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

7.16.1 Detailed Description

L3GD20 Gyroscope API Implementation.

Definition in file gyro.c.

7.17 lib/lowlevel/adc.c File Reference

Analog-to-Digital Converter API Implementation.

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

7.17.1 Detailed Description

Analog-to-Digital Converter API Implementation.

Definition in file adc.c.

7.18 lib/lowlevel/serial.c File Reference

UART Library Implementation.

```
#include <avr/io.h>
#include <avr/interrupt.h>
#include <stdint.h>
#include <stdlib.h>
#include <lowlevel/serial.h>
#include <lowlevel/serial_device.h>
#include <lowlevel/xmem.h>
```

Macros

- #define RX_BUFFER_SIZE 32

  If you define this, a '\r' (CR) will be put in front of a '\n' (LF) when sending a byte.
- #define TX_BUFFER_SIZE 16

  TX Buffer Size in Bytes (Power of 2)
- #define FLOWCONTROL

  Defining this enables incoming XON XOFF (sends XOFF if rx buff is full)
- #define FLOWMARK 5

  Space remaining to trigger xoff/xon.
- #define XON 0x11

  XON Value.
- #define XOFF 0x13

  XOFF Value.

Functions

- uint8_t serialAvailable (void)

  Get number of available UART modules.
- uint8_t serialInit (uint8_t uart, uint16_t baud)

  Initialize the UART Hardware.
- void serialClose (uint8_t uart)

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

  Manually change the flow control.
- uint8_t serialHasChar (uint8_t uart)

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

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

  Read a single byte.
- uint8_t serialRxBufferFull (uint8_t uart)

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

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

  Send a byte.
- void serialWriteString (uint8_t uart, const char ∗data)

  Send a string.
- uint8_t serialTxBufferFull (uint8_t uart)

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

  Check if the transmit buffer is empty.

7.18.1 Detailed Description

UART Library Implementation.

Definition in file serial.c.

7.19 lib/lowlevel/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 <lowlevel/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.

7.19.1 Detailed Description

Time API Implementation.

Definition in file time.c.

7.20 lib/lowlevel/xmem.c File Reference

XMEM API Implementation.

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

7.20.1 Detailed Description

XMEM API Implementation.

Definition in file xmem.c.

7.21 lib/mag.c File Reference

LSM303DLHC Magnetometer API Implementation.

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

Variables

- MagRange magRange

  Stored range to scale returned values.

7.21.1 Detailed Description

LSM303DLHC Magnetometer API Implementation.

Definition in file mag.c.

7.22 lib/remote.c File Reference

Remote API Implementation.

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

Functions

- ISR (TIMER0_OVF_vect)

  External Interrupt detecting Timer0 Overflow, invalidating signal matching.
- ISR (INTn_vect)

  External Interrupt Service Routine for received sum signal.
- void rcInit (void)

  Initialize RC Receiver.

Variables

- volatile int8_t rcValues [RC_CHANNELS]

  Stick positions of remote control.

7.22.1 Detailed Description

Remote API Implementation. Uses Timer0! Base on http://www.rn-wissen.de/index.php/RC--Empfänger_auswerten::C-Programmbeispiel_.28Auslesen_mit_Timer0.29

Definition in file remote.c.

Chapter 8

Example Documentation

8.1 flight.c

```c
/*
 * flight.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 <avr/io.h>
#include <stdlib.h>
#include <avr/interrupt.h>

#include <acc.h>
#include <gyro.h>
#include <mag.h>
#include <remote.h>
#include <lowlevel/adc.h>
#include <lowlevel/serial.h>
#include <lowlevel/time.h>
#include <lowlevel/twi.h>
#include <lowlevel/xmem.h>

int main(void) {

    xmemInit();
    adcInit(AVCC);
    twiInit();
    initSystemTimer();
    serialInit(1, BAUD(38400, F_CPU)); // LCD
    accInit(r4G);
    gyroInit(r2000DPS);
    magInit(r8g1);
    rcInit();

    sei();

    for(;;) {
    }

    return 0;
}
```

Index