Embedded 2 Dokumentation

Erzeugt von Doxygen 1.9.3

Kapitel 1

Dokumentation

1.1 Code Style

Hier wird der aktuelle Coding Style hinterlegt.

1.1.1 EinrÃ¼ckung, Klammern und Formatierung

Der Code sollte mÃ¶glichst einfach lesbar sein und gut strukturiert sein. Die geschwungenen Klammern von BlÃ¶cken wie z.B. If-BlÃ¶cken stehen immer in einer neuen, alleinstehenden Zeile. BlÃ¶cke werden eingerÃ¼ckt.

```
if (x < foo (y, z))
{
  haha = bar[4] + 5;
}
else
{
  while (z)
  {
    haha += foo (z, z);
    z--;
  }
return ++x + bar ();
}
```

Auch bei einzeiligen If-Anweisungen werden geschwungene Klammern verwendet.

1.1.2 Kommentare

Generell sollte jede Variable, Funktion oder andere Logik deren Aufgabe oder Bedeutung nicht direkt erkennbar ist mit einem kurzen Kommentar beschrieben werden. Falls der Kommentar sich Ã¼ber mehrere Zeilen erstreckt, wird dieser mit â/∗ â¦. ∗/â begrenzt. Andernfalls reichen die zwei doppelten Slashs â//â. Kommentare sind generell in Deutsch.

```
int icounter_5; //Counter mit Startwert 5

int rgb_to_lumi(int r, int g, int b).... /*Funktion um RGB-Werte in   einen Luminanz-Wert zu konvertieren*/
```

1.1.3 Bezeichner / Namen von Variablen, Funktionen & Konstanten

Die Namen von den verschiedenen Strukturen sollen schon beim Lesen einen Hinweis auf deren Funktion geben. Die Namen sind generell in Englisch verfasst und kÃ¶nnen auch AbkÃ¼rzungen enthalten. Variablen- und Funktionen-Bezeichner werden durch Unterstriche getrennt und sind klein geschrieben. (snake_case) Konstanten hingegen werden groÃ geschrieben.

```
//Konstanten
int DELAY_MS=4;

//Variablen und Funktionen
int current_memory_left;
void set_all_leds_high();
```

1.1.4 Bibliotheken

Bibliotheken werden generell immer am Anfang der Datei eingebunden.

```
#include "test_3.h"
```

1.1.5 Doxygen

Einleitung mit /∗∗ Nur die nÃ¶tigsten Tags verwenden.

```
/**
* @brief VerzÃ¶gerung (ms)
* VerzÃ¶gerungsfunktion, blockierend
* @param milliseconds Anzahl der zu verzÃ¶gernden Millisekunden
* @return â¦
*/
```

1.2 Programmdokumentation

Die Dokumentation erfolgt gtrennt nach den jeweiligen Aufgaben

1.2.1 Aufgabe I2C Lichtsensor

Es sollen weitere I2C Busteilnehmer an den I2C Bus angeschlossen und angesteuert werden. Für diese Aufgabe zumindest ein I2C Lichtsensor vom Typ BH1750 (Modul GY-302). Sie können gerne auch weitere eigene oder von mir gestellte Sensoren verwenden.

Da der grundliegende I2C Kommunikationsablauf immer ähnlich ist, soll eine universelle FSM entwickelt werden, welche im Interrupt aber auch wahlweise in der Super Loop verwendet werden kann.

Diese wird mittels der Funktion exchangeI2C() getriggert, welche alls Schnittstelle zwischen Anwendungsprogramm und FSM fungiert.

Das Auslesen der Sensor Daten soll mit frei variierbaren Zeitintervallen erfolgen, im Bereich von 1 Sekunde bis 3600 Sekunden. (Makrodefine) Nach dem erfolgreichen Lesen der Sensordaten sollen diese über die UART ausgegeben werden Optional: Erweitern Sie den Kommandointerpreter von Embedded 1 um die Zeitintervalle über die UART verändern zu können.

Kapitel 2

Datenstruktur-Verzeichnis

2.1 Datenstrukturen

Hier folgt die Aufzählung aller Datenstrukturen mit einer Kurzbeschreibung:

Kapitel 3

Datei-Verzeichnis

3.1 Auflistung der Dateien

Hier folgt die Aufzählung aller Dateien mit einer Kurzbeschreibung:

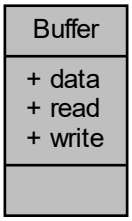
Kapitel 4

Datenstruktur-Dokumentation

4.1 Buffer Strukturreferenz

`#include <UART.h>`

Zusammengehörigkeiten von Buffer:



Datenfelder

- uint8_t data [BUFFER_SIZE]
- uint8_t read
- uint8_t write

4.1.1 Ausführliche Beschreibung

Definiert in Zeile 50 der Datei main_less.c.

4.1.2 Dokumentation der Felder

4.1.2.1 data

`uint8_t data`

Definiert in Zeile 51 der Datei main_less.c.

4.1.2.2 read

`uint8_t read`

Definiert in Zeile 52 der Datei main_less.c.

4.1.2.3 write

`uint8_t write`

Definiert in Zeile 53 der Datei main_less.c.

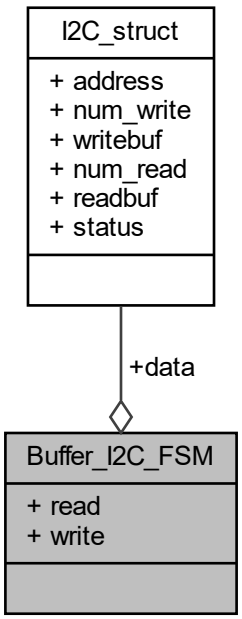
Die Dokumentation für diese Struktur wurde erzeugt aufgrund der Dateien:

- main_less.c
- UART.h

4.2 Buffer_I2C_FSM Strukturreferenz

`#include <I2C.h>`

Zusammengehörigkeiten von Buffer_I2C_FSM:



Datenfelder

- I2C_struct data [BUFFER_SIZE]
- uint8_t read
- uint8_t write

4.2.1 Ausführliche Beschreibung

Definiert in Zeile 35 der Datei I2C.h.

4.2.2 Dokumentation der Felder

4.2.2.1 data

```
I2C_struct data[BUFFER_SIZE]
```

Definiert in Zeile 37 der Datei I2C.h.

4.2.2.2 read

```
uint8_t read
```

Definiert in Zeile 38 der Datei I2C.h.

4.2.2.3 write

```
uint8_t write
```

Definiert in Zeile 39 der Datei I2C.h.

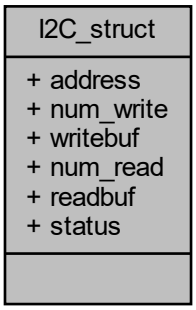
Die Dokumentation für diese Struktur wurde erzeugt aufgrund der Datei:

- I2C.h

4.3 I2C_struct Strukturreferenz

`#include <I2C.h>`

Zusammengehörigkeiten von I2C_struct:



Datenfelder

- uint8_t address
- uint16_t num_write
- uint8_t * writebuf
- uint16_t num_read
- uint8_t * readbuf
- i2c_status_t status

4.3.1 Ausführliche Beschreibung

Definiert in Zeile 24 der Datei I2C.h.

4.3.2 Dokumentation der Felder

4.3.2.1 address

`uint8_t address`

Definiert in Zeile 26 der Datei I2C.h.

4.3.2.2 num_read

```
uint16_t num_read
```

Definiert in Zeile 29 der Datei I2C.h.

4.3.2.3 num_write

```
uint16_t num_write
```

Definiert in Zeile 27 der Datei I2C.h.

4.3.2.4 readbuf

```
uint8_t* readbuf
```

Definiert in Zeile 30 der Datei I2C.h.

4.3.2.5 status

```
i2c_status_t status
```

Definiert in Zeile 31 der Datei I2C.h.

4.3.2.6 writebuf

```
uint8_t* writebuf
```

Definiert in Zeile 28 der Datei I2C.h.

Die Dokumentation für diese Struktur wurde erzeugt aufgrund der Datei:

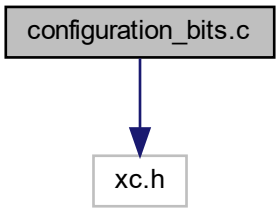
- I2C.h

Kapitel 5

Datei-Dokumentation

5.1 configuration_bits.c-Dateireferenz

`#include <xc.h>`
Include-Abhängigkeitsdiagramm für configuration_bits.c:



5.2 configuration_bits.c

gehe zur Dokumentation dieser Datei

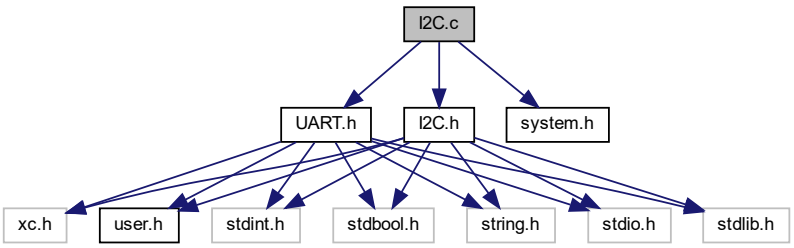
```
00001
00002 // DSPIC33EP512MU810 Configuration Bit Settings
00003
00004 // 'C' source line config statements
00005
00006 // FGS
00007 #pragma config GWRP = OFF               // General Segment Write-Protect bit (General Segment may be
       written)
00008 #pragma config GSS = OFF                // General Segment Code-Protect bit (General Segment Code
       protect is disabled)
00009 #pragma config GSSK = OFF               // General Segment Key bits (General Segment Write Protection
       and Code Protection is Disabled)
00010
00011 // FOSCSEL
00012 #pragma config FNOSC = FRCDIVN          // Initial Oscillator Source Selection Bits (Internal Fast RC
       (FRC) Oscillator with postscaler)
00013 #pragma config IESO = ON                // Two-speed Oscillator Start-up Enable bit (Start up device
       with FRC, then switch to user-selected oscillator source)
00014
00015 // FOSC
```

```
00016 #pragma config POSCMD = XT              // Primary Oscillator Mode Select bits (XT Crystal Oscillator
       Mode)
00017 #pragma config OSCIOFNC = OFF           // OSC2 Pin Function bit (OSC2 is clock output)
00018 #pragma config IOL1WAY = ON             // Peripheral pin select configuration (Allow only one
       reconfiguration)
00019 #pragma config FCKSM = CSECMD           // Clock Switching Mode bits (Clock switching is
       enabled,Fail-safe Clock Monitor is disabled)
00020
00021 // FWDT
00022 #pragma config WDTPOST = PS32768        // Watchdog Timer Postscaler Bits (1:32,768)
00023 #pragma config WDTPRE = PR128           // Watchdog Timer Prescaler bit (1:128)
00024 #pragma config PLLKEN = ON              // PLL Lock Wait Enable bit (Clock switch to PLL source will
       wait until the PLL lock signal is valid.)
00025 #pragma config WINDIS = OFF             // Watchdog Timer Window Enable bit (Watchdog Timer in
       Non-Window mode)
00026 #pragma config FWDTEN = ON              // Watchdog Timer Enable bit (Watchdog timer always enabled)
00027
00028 // FPOR
00029 #pragma config FPWRT = PWR128           // Power-on Reset Timer Value Select bits (128ms)
00030 #pragma config BOREN = ON               // Brown-out Reset (BOR) Detection Enable bit (BOR is enabled)
00031 #pragma config ALTI2C1 = OFF            // Alternate I2C pins for I2C1 (SDA1/SCK1 pins are selected as
       the I/O pins for I2C1)
00032 #pragma config ALTI2C2 = ON            // Alternate I2C pins for I2C2 (SDA2/SCK2 pins are selected as
       the I/O pins for I2C2)
00033
00034 // FICD
00035 #pragma config ICS = PGD1               // ICD Communication Channel Select bits (Communicate on PGEC1
       and PGED1)
00036 #pragma config RSTPRI = PF              // Reset Target Vector Select bit (Device will obtain reset
       instruction from Primary flash)
00037 #pragma config JTAGEN = OFF             // JTAG Enable bit (JTAG is disabled)
00038
00039 // FAS
00040 #pragma config AWRP = OFF               // Auxiliary Segment Write-protect bit (Auxiliary program
       memory is not write-protected)
00041 #pragma config APL = OFF                // Auxiliary Segment Code-protect bit (Aux Flash Code protect
       is disabled)
00042 #pragma config APLK = OFF               // Auxiliary Segment Key bits (Aux Flash Write Protection and
       Code Protection is Disabled)
00043
00044 // #pragma config statements should precede project file includes.
00045 // Use project enums instead of #define for ON and OFF.
00046
00047 #include <xc.h>
00048
00049
00050
00051
```

5.3 I2C.c-Dateireferenz

```
#include "I2C.h"
#include "system.h"
#include "UART.h"
```

Include-Abhängigkeitsdiagramm für I2C.c:



Funktionen

- int16_t put_I2C_struct_FIFO (I2C_struct s)

  Legt eine I2C-Anfrage in dem I2C-FIFO ab.
- int16_t get_I2C_struct_FIFO (volatile I2C_struct *s)

  Entnimmt I2C-Anfrage aus dem I2C-FIFO.
- int16_t exchangeI2C (uint8_t address, uint16_t num_write, uint8_t *writebuf, uint16_t num_read, uint8_t *readbuf, i2c_status_t *status)

  Übergibt angeforderte I2C-Anfrage an das FIFO und liefert den akutellen Status.
- void doI2C ()

  Wird jede ms in der Superloop ausgeführt und beinhalten die FSM für die I2C-Kommunikation.
- void initI2C ()

  Initialisiert die I2C-Kommunikation.
- void * FSM_Idle (void)

  Kopiert die Anfrage aus dem FIFO und leitet Start-Sequenz ein.
- void * FSM_Start (void)

  Beschreibt das Trancieve-Register mit der Adresse.
- void * FSM_Adresse_Write (void)

  Schreibt die zu übertragende Daten in das Tranceive-Register.
- void * FSM_Repeated_Start (void)

  Leitet einen Repeated Start ein und beschreibt das Tranceive Register mit der Adresse.
- void * FSM_Adresse_Read (void)

  Initiiert das Lesen der Daten des Slaves.
- void * FSM_RECV_EN (void)

  Auslesen des Receive Registers und Bestätigung mit ACK bzw.
- void * FSM_Stop (void)

  Überprüfung auf Abschluss der Stop-Sequenz und Rückkehr in den Idle-State.
- void print_sensor_values ()

  Ausgabe der ausgelesenen Sensor-Werte per UART.

5.3.1 Dokumentation der Funktionen

5.3.1.1 doI2C()

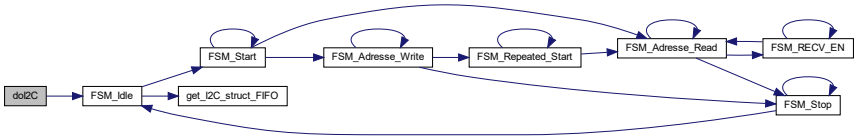
```
void doI2C (
            void  )
```

Wird jede ms in der Superloop ausgeführt und beinhalten die FSM für die I2C-Kommunikation.

Falls das FIFO neue Anfragen enthält wird die FSM getriggert.

Definiert in Zeile 86 der Datei I2C.c.

Hier ist ein Graph, der zeigt, was diese Funktion aufruft:



Hier ist ein Graph der zeigt, wo diese Funktion aufgerufen wird:

5.3.1.2 exchangeI2C()

```
int16_t exchangeI2C (
            uint8_t address,
            uint16_t num_write,
            uint8_t * writebuf,
            uint16_t num_read,
            uint8_t * readbuf,
            i2c_status_t * status )
```

Übergibt angeforderte I2C-Anfrage an das FIFO und liefert den akutellen Status.

Parameter

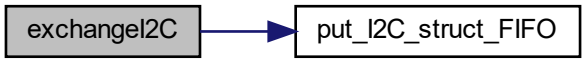
address	Adresse des Slaves
num_write	Anzahl der zu sendenen Bytes
writebuf	Zeiger auf zu schreibende Daten
num_read	Anzahl der zu lesenden Bytes
readbuf	Zeiger auf Berecih, in welchem Daten abgespeuchert werden sollen
status	Zeiger, um akuellen Status zurückzugeben

Rückgabe

1, falls Anfrage direkt angenommen werden konnte, ansonsten 0

Definiert in Zeile 63 der Datei I2C.c.

Hier ist ein Graph, der zeigt, was diese Funktion aufruft:



Hier ist ein Graph der zeigt, wo diese Funktion aufgerufen wird:

5.3.1.3 FSM_Adresse_Read()

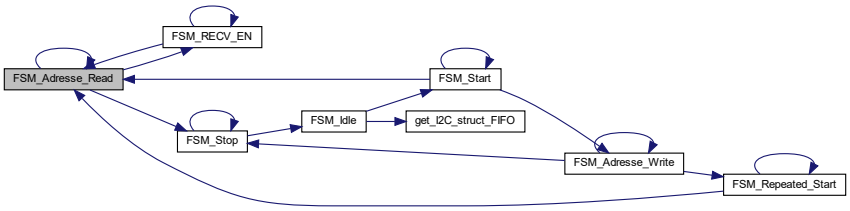
```
void * FSM_Adresse_Read (
            void  )
```

Initiiert das Lesen der Daten des Slaves.

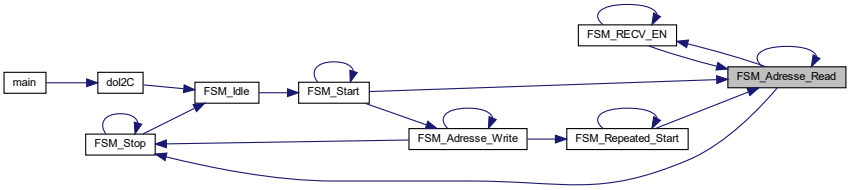
Rückgabe

Definiert in Zeile 236 der Datei I2C.c.

Hier ist ein Graph, der zeigt, was diese Funktion aufruft:



Hier ist ein Graph der zeigt, wo diese Funktion aufgerufen wird:



5.3.1.4 FSM_Adresse_Write()

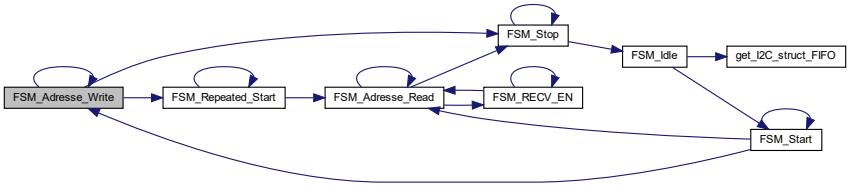
```
void * FSM_Adresse_Write (
            void  )
```

Schreibt die zu übertragende Daten in das Tranceive-Register.

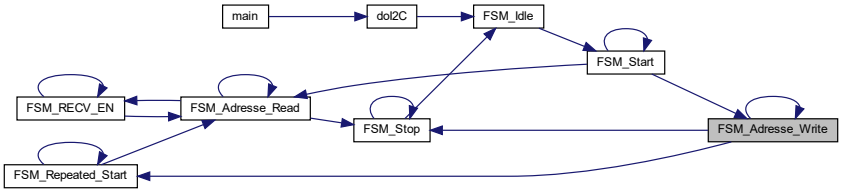
Rückgabe

Definiert in Zeile 179 der Datei I2C.c.

Hier ist ein Graph, der zeigt, was diese Funktion aufruft:



Hier ist ein Graph der zeigt, wo diese Funktion aufgerufen wird:



5.3.1.5 FSM_Idle()

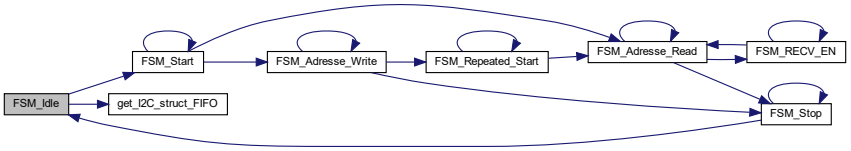
```
void * FSM_Idle (
            void  )
```

Kopiert die Anfrage aus dem FIFO und leitet Start-Sequenz ein.

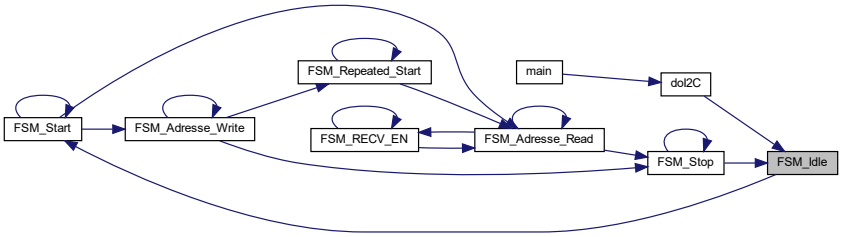
Rückgabe

Definiert in Zeile 140 der Datei I2C.c.

Hier ist ein Graph, der zeigt, was diese Funktion aufruft:



Hier ist ein Graph der zeigt, wo diese Funktion aufgerufen wird:



5.3.1.6 FSM_RECV_EN()

```
void * FSM_RECV_EN (
            void  )
```

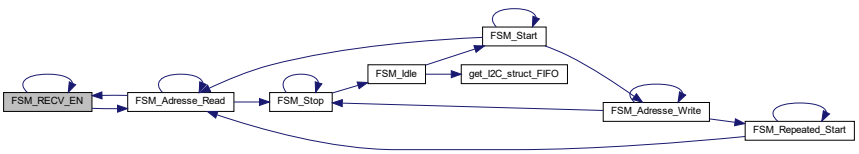
Auslesen des Receive Registers und Bestätigung mit ACK bzw.

NACK

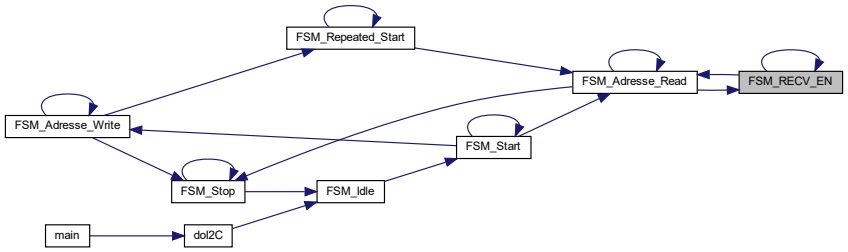
Rückgabe

Definiert in Zeile 284 der Datei I2C.c.

Hier ist ein Graph, der zeigt, was diese Funktion aufruft:



Hier ist ein Graph der zeigt, wo diese Funktion aufgerufen wird:



5.3.1.7 FSM_Repeated_Start()

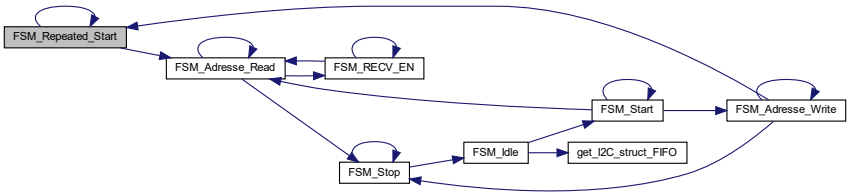
```
void * FSM_Repeated_Start (
            void  )
```

Leitet einen Repeated Start ein und beschreibt das Tranceive Register mit der Adresse.

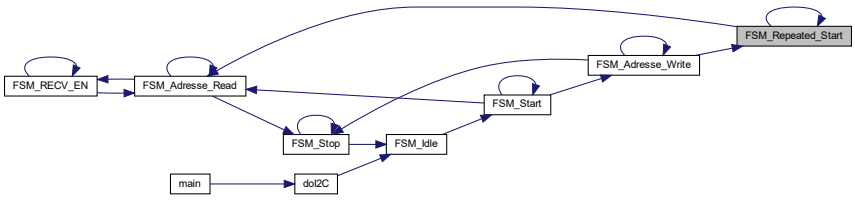
Rückgabe

Definiert in Zeile 220 der Datei I2C.c.

Hier ist ein Graph, der zeigt, was diese Funktion aufruft:



Hier ist ein Graph der zeigt, wo diese Funktion aufgerufen wird:



5.3.1.8 FSM_Start()

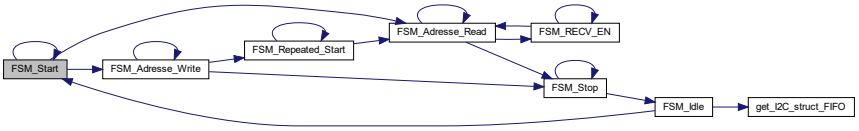
```
void * FSM_Start (
            void  )
```

Beschreibt das Trancieve-Register mit der Adresse.

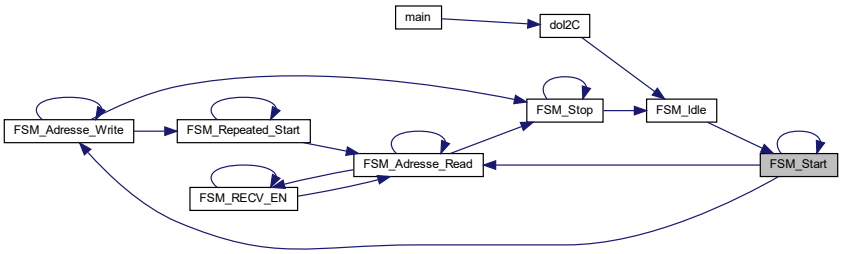
Rückgabe

Definiert in Zeile 152 der Datei I2C.c.

Hier ist ein Graph, der zeigt, was diese Funktion aufruft:



Hier ist ein Graph der zeigt, wo diese Funktion aufgerufen wird:



5.3.1.9 FSM_Stop()

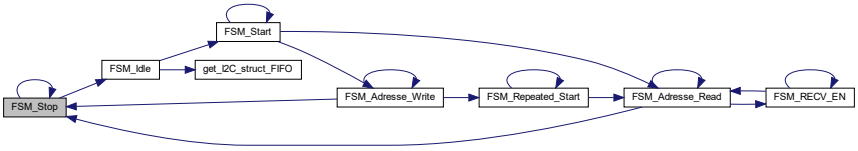
```
void * FSM_Stop (
            void  )
```

Überprüfung auf Abschluss der Stop-Sequenz und Rückkehr in den Idle-State.

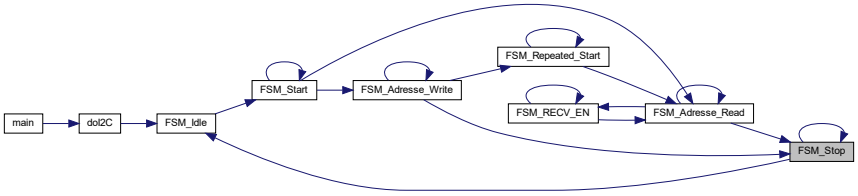
Rückgabe

Definiert in Zeile 314 der Datei I2C.c.

Hier ist ein Graph, der zeigt, was diese Funktion aufruft:



Hier ist ein Graph der zeigt, wo diese Funktion aufgerufen wird:



5.3.1.10 get_I2C_struct_FIFO()

```
int16_t get_I2C_struct_FIFO (
            volatile I2C_struct * s )
```

Entnimmt I2C-Anfrage aus dem I2C-FIFO.

Parameter

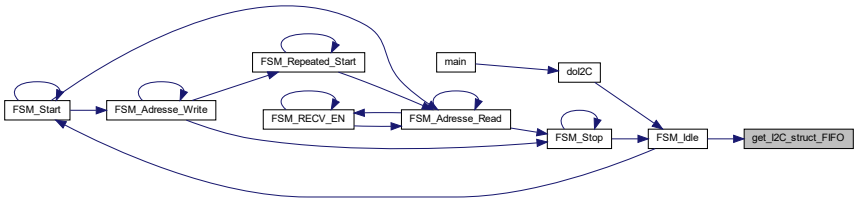
*s	Zeiger auf I2C-Anfrage in Form eines Structs des Typs I2C_struct

Rückgabe

BUFFER_FAIL im Fehlerfall, ansonsten BUFFER_SUCCESS

Definiert in Zeile 35 der Datei I2C.c.

Hier ist ein Graph der zeigt, wo diese Funktion aufgerufen wird:



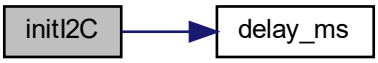
5.3.1.11 initI2C()

```
void initI2C (
              void  )
```

Initialisiert die I2C-Kommunikation.

Definiert in Zeile 105 der Datei I2C.c.

Hier ist ein Graph, der zeigt, was diese Funktion aufruft:



Hier ist ein Graph der zeigt, wo diese Funktion aufgerufen wird:

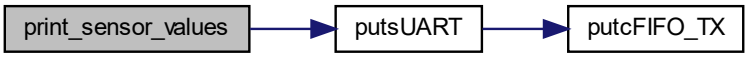
5.3.1.12 print_sensor_values()

```
void print_sensor_values (
            void  )
```

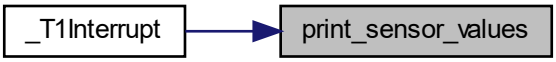
Ausgabe der ausgelesenen Sensor-Werte per UART.

Definiert in Zeile 327 der Datei I2C.c.

Hier ist ein Graph, der zeigt, was diese Funktion aufruft:



Hier ist ein Graph der zeigt, wo diese Funktion aufgerufen wird:



5.3.1.13 put_I2C_struct_FIFO()

```
int16_t put_I2C_struct_FIFO (
            I2C_struct s )
```

Legt eine I2C-Anfrage in dem I2C-FIFO ab.

Parameter

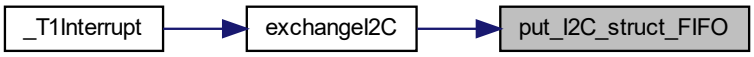
s	I2C-Anfrage in Form eines Structs des Typs I2C_struct

Rückgabe

BUFFER_FAIL im Fehlerfall, ansonsten BUFFER_SUCCESS

Definiert in Zeile 11 der Datei I2C.c.

Hier ist ein Graph der zeigt, wo diese Funktion aufgerufen wird:



5.4 I2C.c

gehe zur Dokumentation dieser Datei

```
00001 #include "I2C.h"
00002 #include "system.h"
00003 #include "UART.h"
00004
00005 //Funktionen
00011 int16_t put_I2C_struct_FIFO(I2C_struct s)
00012 {
00013   if ( ( FIFO_I2C.write + 1 == FIFO_I2C.read ) ||
00014        ( FIFO_I2C.read == 0 && FIFO_I2C.write + 1 == BUFFER_SIZE ) )
00015   {
00016     return BUFFER_FAIL; // voll
00017   }
00018
00019   FIFO_I2C.data[FIFO_I2C.write] = s;
00020
00021   FIFO_I2C.write++;
00022   if (FIFO_I2C.write >= BUFFER_SIZE)
00023   {
00024     FIFO_I2C.write = 0;
00025   }
00026   return BUFFER_SUCCESS;
00027 }
00028
00029
00035 int16_t get_I2C_struct_FIFO(volatile I2C_struct *s)
00036 {
00037   if (FIFO_I2C.read == FIFO_I2C.write)
00038   {
00039     return BUFFER_FAIL;
00040   }
00041   *s = FIFO_I2C.data[FIFO_I2C.read];
00042
00043   FIFO_I2C.read++;
00044   if (FIFO_I2C.read >= BUFFER_SIZE)
00045   {
00046     FIFO_I2C.read = 0;
00047   }
00048   return BUFFER_SUCCESS;
00049 }
00050
00051
00063 int16_t exchangeI2C(uint8_t address, uint16_t num_write, uint8_t *writebuf, uint16_t num_read, uint8_t
       *readbuf, i2c_status_t *status)
00064 {
00065     I2C_struct temporary_struct = {address,num_write,writebuf,num_read,readbuf,Pending};
00066     put_I2C_struct_FIFO(temporary_struct);
00067
00068     *status = I2C_test_struct.status;
00069
00070     if (I2C_test_struct.status==Finished)
00071     {
00072
00073         return 1;
00074     }
00075     else
00076     {
00077         return 0;
00078     }
00079 }
00080
00086 void doI2C()
00087 {
```

```
00088   static StateFunc statefunc = FSM_Idle;
00089
00090   if (!(FIFO_I2C.read == FIFO_I2C.write)) //Wenn Ihnalt im FIFO ist
00091   {
00092       trigger_FSM=1;
00093   }
00094
00095   if (trigger_FSM==1)
00096   {
00097       statefunc = (StateFunc)(*statefunc)();
00098   }
00099
00100 }
00101
00105 void initI2C()
00106 {
00107     I2C2CONbits.A10M = 0;
00108     I2C2BRG = 245; //100kHz
00109
00110     // Einschalten I2C mit eigenem Workaround, I2C Peripheriemodul kann hier leider nicht verwendet
      werden
00111     I2C_SDA_TRIS = 1;    // Pins wie einen Open-Kollektor-Treiber verwenden, d.h. 1 - hochohmig, 0
      wird getrieben
00112     I2C_SCL_TRIS = 1;
00113     I2C_SDA = 0;
00114     I2C_SCL = 0;
00115
00116         int j;
00117         for (j=0; j<=9; j++)    // takten bis min 1 Byte
00118         {
00119           I2C_SCL_TRIS = 0; delay_ms(1); // 5 us wÃ¤ren ausreichend ...100 kBaud
00120           I2C_SCL_TRIS = 1; delay_ms(1);
00121         }
00122         // Start Condition senden
00123         I2C_SCL_TRIS = 0;  delay_ms(1);
00124         I2C_SDA_TRIS = 0;  delay_ms(1);
00125         // Stop Condition senden
00126         I2C_SCL_TRIS = 1;  delay_ms(1);
00127         I2C_SDA_TRIS = 1;  delay_ms(1);
00128
00129         // Nun I2C erst anschalten
00130         _MI2C2IF = 0; //Interrupt falls noetig
00131         _MI2C2IE = 0;
00132         I2C2CONbits.I2CEN = 1;
00133 }
00134
00135
00140 void *FSM_Idle(void)
00141 {
00142     get_I2C_struct_FIFO(&I2C_test_struct);
00143     I2C2CONbits.SEN=1; //Start
00144     return FSM_Start;
00145
00146 }
00147
00152 void *FSM_Start(void)
00153 {
00154
00155     if (I2C2CONbits.SEN==0)
00156     {
00157         if (I2C_test_struct.num_write>0) //Schreiben
00158         {
00159             I2C2TRN=(I2C_test_struct.address<<1);
00160             return FSM_Adresse_Write;
00161         }
00162
00163         else if (I2C_test_struct.num_read>0)  //Lesen
00164         {
00165             I2C2TRN=(I2C_test_struct.address<<1) | 0b1;
00166             return FSM_Adresse_Read;
00167         }
00168
00169
00170     }
00171     return FSM_Start;
00172
00173 }
00174
00179 void *FSM_Adresse_Write(void)
00180 {
00181     if(I2C2STATbits.TRSTAT==0) //Wenn erfolgreich übertragen
00182     {
00183         if (I2C2STATbits.ACKSTAT==1) //if NACK received, generate stop condition and exit
00184         {
00185             I2C2CONbits.PEN=1;
00186             I2C_test_struct.status=Error;
00187             return FSM_Stop;
```

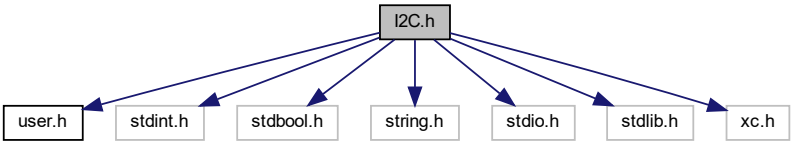
```
00188         }
00189
00190         if (I2C2STATbits.ACKSTAT==0) //ACK von Slave erhalten
00191         {
00192             static int count=0;
00193
00194             if (count < I2C_test_struct.num_write) //Noch Bytes zu senden
00195             {
00196
00197                 I2C2TRN=I2C_test_struct.writebuf[count];
00198                 count++;
00199                 return FSM_Adresse_Write;
00200             }
00201
00202             else //Nichts mehr zu schicken
00203             {
00204                count=0;
00205                I2C2CONbits.RSEN=1;
00206                return FSM_Repeated_Start;
00207             }
00208
00209         }
00210
00211     }
00212     return FSM_Adresse_Write;
00213
00214 }
00215
00220 void *FSM_Repeated_Start(void)
00221 {
00222     if (I2C2CONbits.RSEN==0)
00223     {
00224         I2C2TRN=(I2C_test_struct.address«1) | 0b1;
00225         return FSM_Adresse_Read;
00226     }
00227     return FSM_Repeated_Start;
00228 }
00229
00230
00236 void *FSM_Adresse_Read(void)
00237 {
00238     if(I2C2STATbits.TRSTAT==0) //Wenn erfolgreich übertragen
00239     {
00240         if (I2C2STATbits.ACKSTAT==1) //if NACK received, generate stop condition and exit
00241         {
00242             I2C2CONbits.PEN=1;
00243             I2C_test_struct.status=Error;
00244             return FSM_Stop;
00245         }
00246
00247         if (I2C2STATbits.ACKSTAT==0)
00248         {
00249             if (I2C2CONbits.ACKEN==0)
00250             {
00251                 static int count = 0;
00252
00253                 if (count < I2C_test_struct.num_read) //Noch Bytes zu empfangen
00254                 {
00255                     count++;
00256                     I2C2CONbits.RCEN=1;
00257                     return FSM_RECV_EN;
00258                 }
00259
00260                 else //Nichts mehr zu empfangen
00261                 {
00262                    count = 0;
00263                    I2C2CONbits.PEN=1;
00264                    I2C_test_struct.status=Finished;
00265                    return FSM_Stop;
00266                 }
00267             }
00268             else
00269             {
00270                 return FSM_Adresse_Read;
00271             }
00272
00273         }
00274
00275     }
00276     return FSM_Adresse_Read;
00277
00278 }
00279
00284 void *FSM_RECV_EN(void)
00285 {
00286     if (I2C2CONbits.RCEN==0)
00287     {
```

```
00288         static int count = 0;
00289         I2C_test_struct.readbuf[count]=I2C2RCV;
00290         count++;
00291 
00292         if (count>=I2C_test_struct.num_read) //Wenn letztes Byte empfangen wurde
00293         {
00294             count=0;
00295             I2C2CONbits.ACKDT=1; //NACK
00296         }
00297         else
00298         {
00299             I2C2CONbits.ACKDT=0; //ACK
00300         }
00301 
00302         I2C2CONbits.ACKEN=1;
00303         return FSM_Adresse_Read;
00304     }
00305 
00306     return FSM_RECV_EN;
00307 
00308 }
00309 
00314 void *FSM_Stop(void)
00315 {
00316     if(I2C2CONbits.PEN==0)
00317     {
00318         trigger_FSM=0;
00319         return FSM_Idle;
00320     }
00321     return FSM_Stop;
00322 }
00323 
00327 void print_sensor_values()
00328 {
00329     //Temperatur
00330     double temp = read_data_buffer_temp[0]<<8|read_data_buffer_temp[1];
00331     char str[16];
00332     sprintf(str,"%.1f",temp/256);
00333     putsUART("Temperatur: ");
00334     putsUART(str);
00335     putsUART("°C");
00336     putsUART("\n");
00337 
00338     //Licht
00339     double light = read_data_buffer_light[0]<<8 | read_data_buffer_light[1];
00340 
00341     sprintf(str,"%.1f",light/1.2);
00342     putsUART("Licht: ");
00343     putsUART(str);
00344     putsUART(" lux");
00345     putsUART("\n");
00346 
00347 }
```

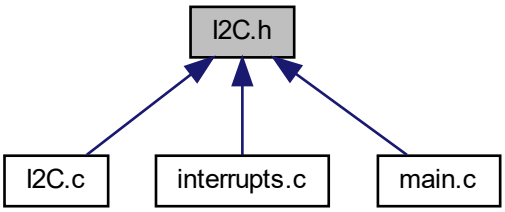
5.5 I2C.h-Dateireferenz

```
#include "user.h"
#include <stdint.h>
#include <stdbool.h>
#include <string.h>
#include <stdio.h>
#include <stdlib.h>
#include <xc.h>
```

Include-Abhängigkeitsdiagramm für I2C.h:



Dieser Graph zeigt, welche Datei direkt oder indirekt diese Datei enthält:



Datenstrukturen

- struct I2C_struct
- struct Buffer_I2C_FSM

Makrodefinitionen

- #define I2C_SCL _RA2

  In dieser Header-Datei werden alle für das I2C-Protokoll benötigten Konstanten, Typedefs und Prototypen definiert.
- #define I2C_SDA _RA3
- #define I2C_SCL_TRIS _TRISA2
- #define I2C_SDA_TRIS _TRISA3

Typdefinitionen

- typedef void ∗(∗ StateFunc) ()

Aufzählungen

- enum i2c_status_t { Pending , Finished , Error }

Funktionen

- int16_t exchangeI2C (uint8_t address, uint16_t num_write, uint8_t *writebuf, uint16_t num_read, uint8_t *readbuf, i2c_status_t *status)

  Übergibt angeforderte I2C-Anfrage an das FIFO und liefert den akutellen Status.
- void doI2C (void)

  Wird jede ms in der Superloop ausgeführt und beinhalten die FSM für die I2C-Kommunikation.
- void initI2C (void)

  Initialisiert die I2C-Kommunikation.
- void print_sensor_values (void)

  Ausgabe der ausgelesenen Sensor-Werte per UART.
- void * FSM_Idle (void)

  Kopiert die Anfrage aus dem FIFO und leitet Start-Sequenz ein.
- void * FSM_Start (void)

  Beschreibt das Trancieve-Register mit der Adresse.
- void * FSM_Adresse_Read (void)

  Initiiert das Lesen der Daten des Slaves.
- void * FSM_Adresse_Write (void)

  Schreibt die zu übertragende Daten in das Tranceive-Register.
- void * FSM_Repeated_Start (void)

  Leitet einen Repeated Start ein und beschreibt das Tranceive Register mit der Adresse.
- void * FSM_RECV_EN (void)

  Auslesen des Receive Registers und Bestätigung mit ACK bzw.
- void * FSM_Stop (void)

  Überprüfung auf Abschluss der Stop-Sequenz und Rückkehr in den Idle-State.

Variablen

- uint8_t write_data_buffer_temp
- uint8_t write_data_buffer_light
- uint8_t read_data_buffer_temp [2]
- uint8_t read_data_buffer_light [2]
- bool trigger_FSM
- I2C_struct I2C_test_struct
- Buffer_I2C_FSM FIFO_I2C

5.5.1 Makro-Dokumentation

5.5.1.1 I2C_SCL

```
#define I2C_SCL _RA2
```

In dieser Header-Datei werden alle für das I2C-Protokoll benötigten Konstanten, Typedefs und Prototypen definiert.

Definiert in Zeile 16 der Datei I2C.h.

5.5.1.2 I2C_SCL_TRIS

```
#define I2C_SCL_TRIS _TRISA2
```

Definiert in Zeile 18 der Datei I2C.h.

5.5.1.3 I2C_SDA

```
#define I2C_SDA _RA3
```

Definiert in Zeile 17 der Datei I2C.h.

5.5.1.4 I2C_SDA_TRIS

```
#define I2C_SDA_TRIS _TRISA3
```

Definiert in Zeile 19 der Datei I2C.h.

5.5.2 Dokumentation der benutzerdefinierten Typen

5.5.2.1 StateFunc

```
typedef void *(* StateFunc) ()
```

Definiert in Zeile 42 der Datei I2C.h.

5.5.3 Dokumentation der Aufzählungstypen

5.5.3.1 i2c_status_t

```
enum i2c_status_t
```

Aufzählungswerte

Pending	
Finished	
Error	

Definiert in Zeile 22 der Datei I2C.h.

5.5.4 Dokumentation der Funktionen

5.5.4.1 doI2C()

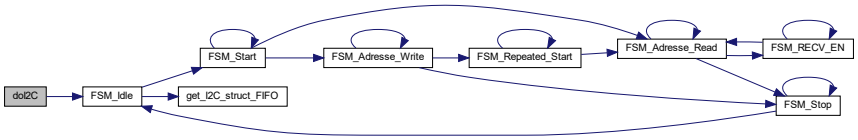
```
void doI2C (
            void  )
```

Wird jede ms in der Superloop ausgeführt und beinhalten die FSM für die I2C-Kommunikation.

Falls das FIFO neue Anfragen enthält wird die FSM getriggert.

Definiert in Zeile 86 der Datei I2C.c.

Hier ist ein Graph, der zeigt, was diese Funktion aufruft:



Hier ist ein Graph der zeigt, wo diese Funktion aufgerufen wird:

5.5.4.2 exchangeI2C()

```
int16_t exchangeI2C (
            uint8_t address,
            uint16_t num_write,
            uint8_t * writebuf,
            uint16_t num_read,
            uint8_t * readbuf,
            i2c_status_t * status )
```

Übergibt angeforderte I2C-Anfrage an das FIFO und liefert den akutellen Status.

Parameter

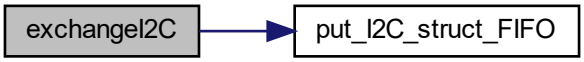
address	Adresse des Slaves
num_write	Anzahl der zu sendenen Bytes
writebuf	Zeiger auf zu schreibende Daten
num_read	Anzahl der zu lesenden Bytes
readbuf	Zeiger auf Berecih, in welchem Daten abgespeuchert werden sollen
status	Zeiger, um akuellen Status zurückzugeben

Rückgabe

1, falls Anfrage direkt angenommen werden konnte, ansonsten 0

Definiert in Zeile 63 der Datei I2C.c.

Hier ist ein Graph, der zeigt, was diese Funktion aufruft:



Hier ist ein Graph der zeigt, wo diese Funktion aufgerufen wird:

5.5.4.3 FSM_Adresse_Read()

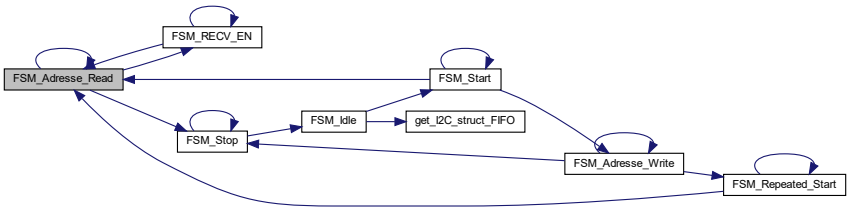
```
void * FSM_Adresse_Read (
            void  )
```

Initiiert das Lesen der Daten des Slaves.

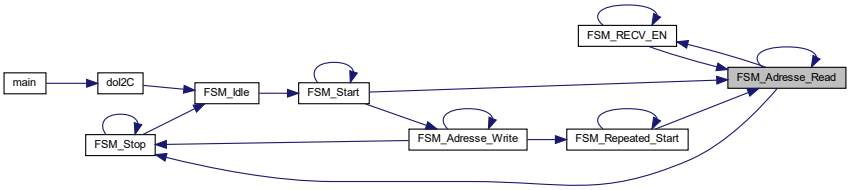
Rückgabe

Definiert in Zeile 236 der Datei I2C.c.

Hier ist ein Graph, der zeigt, was diese Funktion aufruft:



Hier ist ein Graph der zeigt, wo diese Funktion aufgerufen wird:



5.5.4.4 FSM_Adresse_Write()

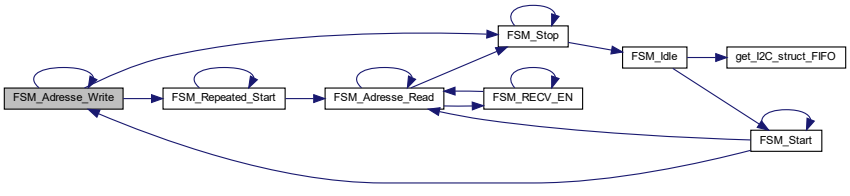
```
void * FSM_Adresse_Write (
            void  )
```

Schreibt die zu übertragende Daten in das Tranceive-Register.

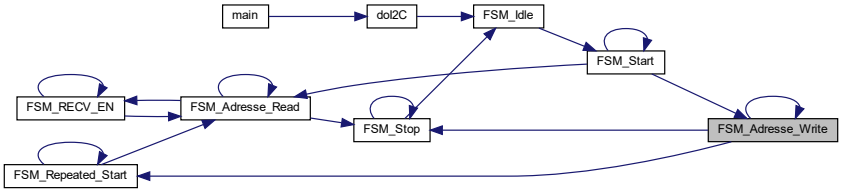
Rückgabe

Definiert in Zeile 179 der Datei I2C.c.

Hier ist ein Graph, der zeigt, was diese Funktion aufruft:



Hier ist ein Graph der zeigt, wo diese Funktion aufgerufen wird:



5.5.4.5 FSM_Idle()

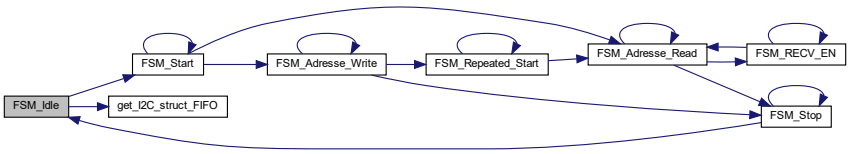
```
void * FSM_Idle (
            void  )
```

Kopiert die Anfrage aus dem FIFO und leitet Start-Sequenz ein.

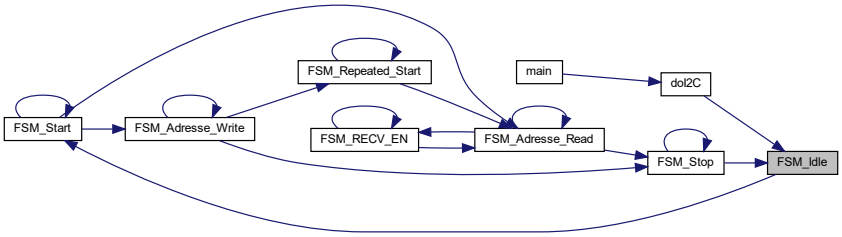
Rückgabe

Definiert in Zeile 140 der Datei I2C.c.

Hier ist ein Graph, der zeigt, was diese Funktion aufruft:



Hier ist ein Graph der zeigt, wo diese Funktion aufgerufen wird:



5.5.4.6 FSM_RECV_EN()

```
void * FSM_RECV_EN (
            void  )
```

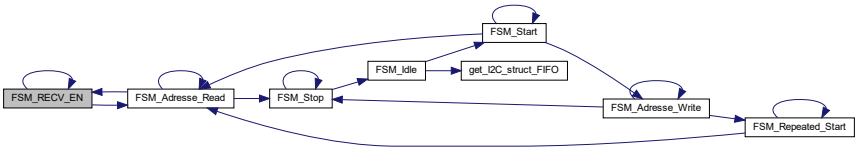
Auslesen des Receive Registers und Bestätigung mit ACK bzw.

NACK

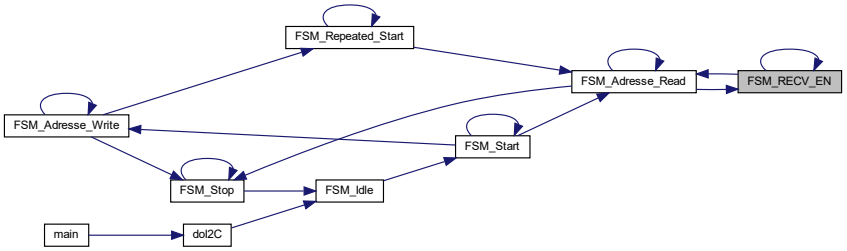
Rückgabe

Definiert in Zeile 284 der Datei I2C.c.

Hier ist ein Graph, der zeigt, was diese Funktion aufruft:



Hier ist ein Graph der zeigt, wo diese Funktion aufgerufen wird:



5.5.4.7 FSM_Repeated_Start()

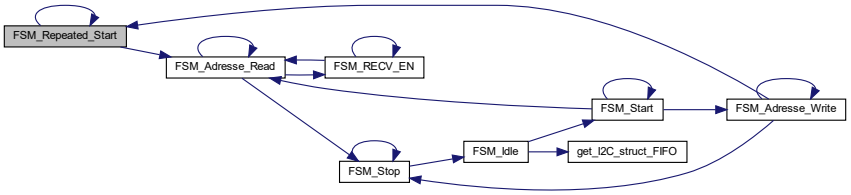
```
void * FSM_Repeated_Start (
            void  )
```

Leitet einen Repeated Start ein und beschreibt das Tranceive Register mit der Adresse.

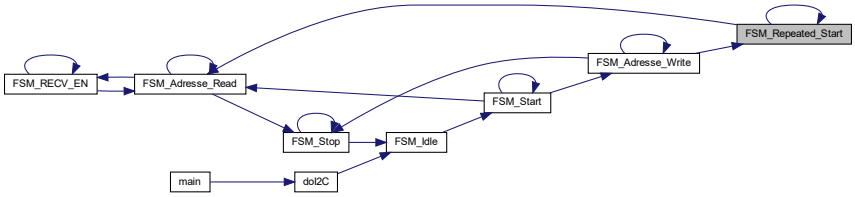
Rückgabe

Definiert in Zeile 220 der Datei I2C.c.

Hier ist ein Graph, der zeigt, was diese Funktion aufruft:



Hier ist ein Graph der zeigt, wo diese Funktion aufgerufen wird:



5.5.4.8 FSM_Start()

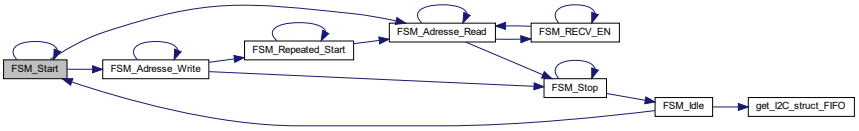
```
void * FSM_Start (
            void  )
```

Beschreibt das Trancieve-Register mit der Adresse.

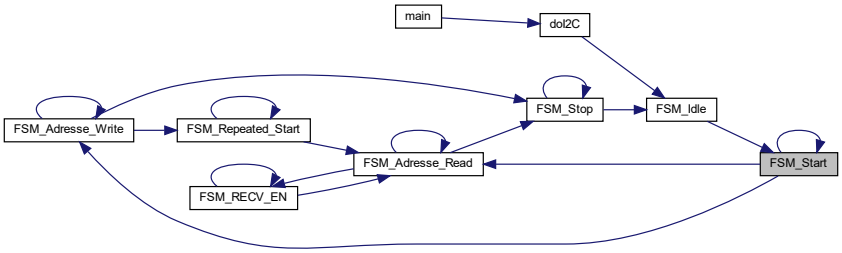
Rückgabe

Definiert in Zeile 152 der Datei I2C.c.

Hier ist ein Graph, der zeigt, was diese Funktion aufruft:



Hier ist ein Graph der zeigt, wo diese Funktion aufgerufen wird:



5.5.4.9 FSM_Stop()

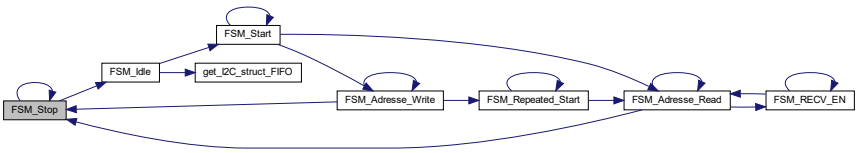
```
void * FSM_Stop (
            void  )
```

Überprüfung auf Abschluss der Stop-Sequenz und Rückkehr in den Idle-State.

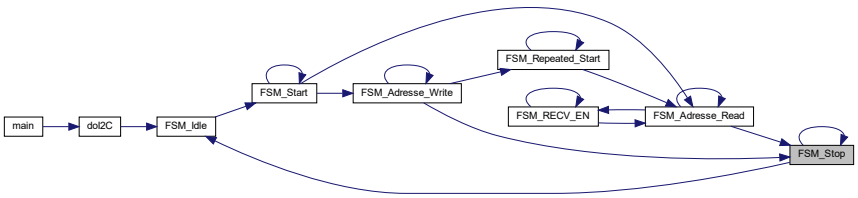
Rückgabe

Definiert in Zeile 314 der Datei I2C.c.

Hier ist ein Graph, der zeigt, was diese Funktion aufruft:



Hier ist ein Graph der zeigt, wo diese Funktion aufgerufen wird:



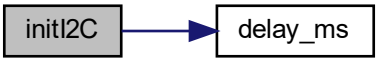
5.5.4.10 initI2C()

```
void initI2C (
             void  )
```

Initialisiert die I2C-Kommunikation.

Definiert in Zeile 105 der Datei I2C.c.

Hier ist ein Graph, der zeigt, was diese Funktion aufruft:



Hier ist ein Graph der zeigt, wo diese Funktion aufgerufen wird:

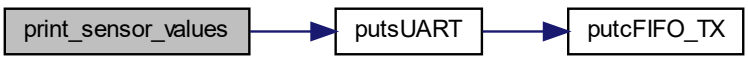
5.5.4.11 print_sensor_values()

```
void print_sensor_values (
             void  )
```

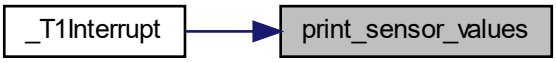
Ausgabe der ausgelesenen Sensor-Werte per UART.

Definiert in Zeile 327 der Datei I2C.c.

Hier ist ein Graph, der zeigt, was diese Funktion aufruft:



Hier ist ein Graph der zeigt, wo diese Funktion aufgerufen wird:



5.5.5 Variablen-Dokumentation

5.5.5.1 FIFO_I2C

```
Buffer_I2C_FSM FIFO_I2C  [extern]
```

5.5.5.2 I2C_test_struct

```
I2C_struct I2C_test_struct  [extern]
```

5.5.5.3 read_data_buffer_light

```
uint8_t read_data_buffer_light[2]  [extern]
```

5.5.5.4 read_data_buffer_temp

```
uint8_t read_data_buffer_temp[2]  [extern]
```

5.5.5.5 trigger_FSM

```
bool trigger_FSM  [extern]
```

5.5.5.6 write_data_buffer_light

```
uint8_t write_data_buffer_light  [extern]
```

5.5.5.7 write_data_buffer_temp

```
uint8_t write_data_buffer_temp  [extern]
```

5.6 I2C.h

gehe zur Dokumentation dieser Datei

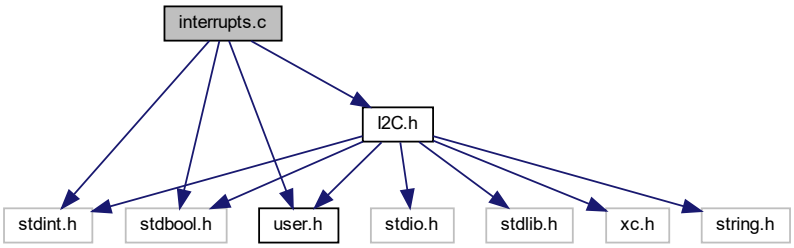
```
00001
00005 #include "user.h"
00006 //#include "UART.h"
00007 #include <stdint.h>        /* Includes uint16_t definition                    */
00008 #include <stdbool.h>       /* Includes true/false definition                  */
00009 #include <string.h>
00010 #include <stdio.h>
00011 #include <stdlib.h>
00012
00013 #include <xc.h>
00014
00015 //Konstanten
00016 #define I2C_SCL      _RA2
00017 #define I2C_SDA      _RA3
00018 #define I2C_SCL_TRIS _TRISA2
00019 #define I2C_SDA_TRIS _TRISA3
00020
00021 //Typedef
00022 typedef enum {Pending, Finished, Error} i2c_status_t;
00023
00024 typedef struct
00025 {
00026     uint8_t address;
00027     uint16_t num_write;
00028     uint8_t *writebuf;
00029     uint16_t num_read;
00030     uint8_t *readbuf;
00031     i2c_status_t status;
00032 }I2C_struct;
00033
00034
00035 typedef struct
00036 {
00037   I2C_struct data[BUFFER_SIZE];
00038   uint8_t read; // zeigt auf das Feld mit dem ältesten Inhalt
00039   uint8_t write; // zeigt immer auf leeres Feld
00040 }Buffer_I2C_FSM;
00041
00042 typedef void *(*StateFunc)();
00043
00044 #ifdef MAIN
00045 //Globale Variablen
00046 uint8_t write_data_buffer_temp;
00047 uint8_t write_data_buffer_light;
00048 uint8_t read_data_buffer_temp[2];
00049 uint8_t read_data_buffer_light[2];
00050
00051 bool trigger_FSM;
00052
00053
00054 I2C_struct I2C_test_struct = {0,0,NULL,0,NULL,Finished};
00055
00056 Buffer_I2C_FSM FIFO_I2C = {{},0,0}; //FIFO für die I2C FSM
00057 #else
00058 extern uint8_t write_data_buffer_temp;
00059 extern uint8_t write_data_buffer_light;
00060 extern uint8_t read_data_buffer_temp[2];
00061 extern uint8_t read_data_buffer_light[2];
00062
00063 extern bool trigger_FSM;
00064
```

```
00065
00066 extern I2C_struct I2C_test_struct;
00067
00068 extern Buffer_I2C_FSM FIFO_I2C; //FIFO für die I2C FSM
00069 #endif
00070
00071
00072
00073
00074 //Prototypen
00075 int16_t exchangeI2C(uint8_t address, uint16_t num_write, uint8_t *writebuf, uint16_t num_read, uint8_t
       *readbuf, i2c_status_t *status);
00076
00077 void doI2C(void);
00078
00079 void initI2C(void);
00080
00081 void print_sensor_values(void);
00082
00083 void *FSM_Idle(void);
00084 void *FSM_Start(void);
00085 void *FSM_Adresse_Read(void);
00086 void *FSM_Adresse_Write(void);
00087 void *FSM_Repeated_Start(void);
00088 void *FSM_RECV_EN(void);
00089
00090 void *FSM_Stop(void);
00091
00092
```

5.7 interrupts.c-Dateireferenz

```
#include <stdint.h>
#include <stdbool.h>
#include "user.h"
#include "I2C.h"
```

Include-Abhängigkeitsdiagramm für interrupts.c:



Funktionen

- void _T1Interrupt (void)

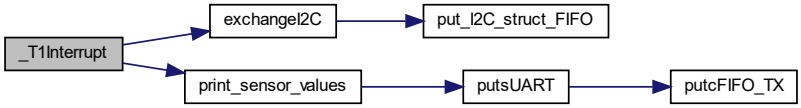
5.7.1 Dokumentation der Funktionen

5.7.1.1 _T1Interrupt()

```
void _T1Interrupt (
            void  )
```

Definiert in Zeile 128 der Datei interrupts.c.

Hier ist ein Graph, der zeigt, was diese Funktion aufruft:



5.8 interrupts.c

gehe zur Dokumentation dieser Datei

```
00001
00002 /* Files to Include                                                 */
00003
00005 /* Device header file */
00006 #if defined(__XC16__)
00007     #include <xc.h>
00008 #elif defined(__C30__)
00009     #if defined(__dsPIC33E__)
00010         #include <p33Exxxx.h>
00011     #elif defined(__dsPIC33F__)
00012         #include <p33Fxxxx.h>
00013     #endif
00014 #endif
00015
00016 #include <stdint.h>        /* Includes uint16_t definition   */
00017 #include <stdbool.h>       /* Includes true/false definition */
00018 #include "user.h"
00019 #include "I2C.h"
00020
00021
00022 /* Interrupt Vector Options                                          */
00023
00024 /*                                                                   */
00025 /* Refer to the C30 (MPLAB C Compiler for PIC24F MCUs and dsPIC33F DSCs) User */
00026 /* Guide for an up to date list of the available interrupt options.  */
00027 /* Alternately these names can be pulled from the device linker scripts.  */
00028 /*                                                                   */
00029 /* dsPIC33F Primary Interrupt Vector Names:                          */
00030 /*                                                                   */
00031 /* _INT0Interrupt      _C1Interrupt                                  */
00032 /* _IC1Interrupt       _DMA3Interrupt                                */
00033 /* _OC1Interrupt       _IC3Interrupt                                 */
00034 /* _T1Interrupt        _IC4Interrupt                                 */
00035 /* _DMA0Interrupt      _IC5Interrupt                                 */
00036 /* _IC2Interrupt       _IC6Interrupt                                 */
00037 /* _OC2Interrupt       _OC5Interrupt                                 */
00038 /* _T2Interrupt        _OC6Interrupt                                 */
00039 /* _T3Interrupt        _OC7Interrupt                                 */
00040 /* _SPI1ErrInterrupt   _OC8Interrupt                                 */
00041 /* _SPI1Interrupt      _DMA4Interrupt                                */
00042 /* _U1RXInterrupt      _T6Interrupt                                  */
00043 /* _U1TXInterrupt      _T7Interrupt                                  */
00044 /* _ADC1Interrupt      _SI2C2Interrupt                               */
00045 /* _DMA1Interrupt      _MI2C2Interrupt                               */
00046 /* _SI2C1Interrupt     _T8Interrupt                                  */
00047 /* _MI2C1Interrupt     _T9Interrupt                                  */
00048 /* _CNInterrupt        _INT3Interrupt                                */
00049 /* _INT1Interrupt      _INT4Interrupt                                */
00050 /* _ADC2Interrupt      _C2RxRdyInterrupt                             */
00051 /* _DMA2Interrupt      _C2Interrupt                                  */
00052 /* _OC3Interrupt       _DCIErrInterrupt                              */
```

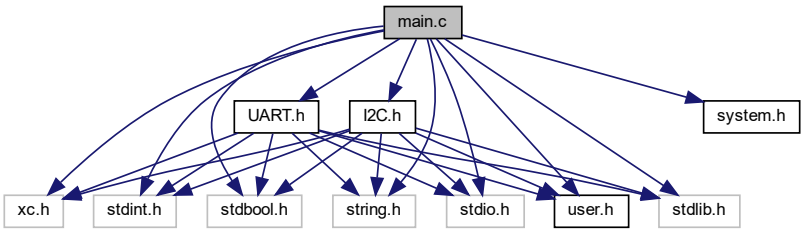
```
00053 /* _OC4Interrupt       _DCIInterrupt                                          */
00054 /* _T4Interrupt        _DMA5Interrupt                                         */
00055 /* _T5Interrupt        _U1ErrInterrupt                                        */
00056 /* _INT2Interrupt      _U2ErrInterrupt                                        */
00057 /* _U2RXInterrupt      _DMA6Interrupt                                         */
00058 /* _U2TXInterrupt      _DMA7Interrupt                                         */
00059 /* _SPI2ErrInterrupt   _C1TxReqInterrupt                                      */
00060 /* _SPI2Interrupt      _C2TxReqInterrupt                                      */
00061 /* _C1RxRdyInterrupt                                                          */
00062 /*                                                                            */
00063 /* dsPIC33E Primary Interrupt Vector Names:                                   */
00064 /*                                                                            */
00065 /* _INT0Interrupt     _IC4Interrupt      _U4TXInterrupt                       */
00066 /* _IC1Interrupt      _IC5Interrupt      _SPI3ErrInterrupt                    */
00067 /* _OC1Interrupt      _IC6Interrupt      _SPI3Interrupt                       */
00068 /* _T1Interrupt       _OC5Interrupt      _OC9Interrupt                        */
00069 /* _DMA0Interrupt     _OC6Interrupt      _IC9Interrupt                        */
00070 /* _IC2Interrupt      _OC7Interrupt      _PWM1Interrupt                       */
00071 /* _OC2Interrupt      _OC8Interrupt      _PWM2Interrupt                       */
00072 /* _T2Interrupt       _PMPInterrupt      _PWM3Interrupt                       */
00073 /* _T3Interrupt       _DMA4Interrupt     _PWM4Interrupt                       */
00074 /* _SPI1ErrInterrupt  _T6Interrupt       _PWM5Interrupt                       */
00075 /* _SPI1Interrupt     _T7Interrupt       _PWM6Interrupt                       */
00076 /* _U1RXInterrupt     _SI2C2Interrupt    _PWM7Interrupt                       */
00077 /* _U1TXInterrupt     _MI2C2Interrupt    _DMA8Interrupt                       */
00078 /* _AD1Interrupt      _T8Interrupt       _DMA9Interrupt                       */
00079 /* _DMA1Interrupt     _T9Interrupt       _DMA10Interrupt                      */
00080 /* _NVMInterrupt      _INT3Interrupt     _DMA11Interrupt                      */
00081 /* _SI2C1Interrupt    _INT4Interrupt     _SPI4ErrInterrupt                    */
00082 /* _MI2C1Interrupt    _C2RxRdyInterrupt  _SPI4Interrupt                       */
00083 /* _CM1Interrupt      _C2Interrupt       _OC10Interrupt                       */
00084 /* _CNInterrupt       _QEI1Interrupt     _IC10Interrupt                       */
00085 /* _INT1Interrupt     _DCIEInterrupt     _OC11Interrupt                       */
00086 /* _AD2Interrupt      _DCIInterrupt      _IC11Interrupt                       */
00087 /* _IC7Interrupt      _DMA5Interrupt     _OC12Interrupt                       */
00088 /* _IC8Interrupt      _RTCCInterrupt     _IC12Interrupt                       */
00089 /* _DMA2Interrupt     _U1ErrInterrupt    _DMA12Interrupt                      */
00090 /* _OC3Interrupt      _U2ErrInterrupt    _DMA13Interrupt                      */
00091 /* _OC4Interrupt      _CRCInterrupt      _DMA14Interrupt                      */
00092 /* _T4Interrupt       _DMA6Interrupt     _OC13Interrupt                       */
00093 /* _T5Interrupt       _DMA7Interrupt     _IC13Interrupt                       */
00094 /* _INT2Interrupt     _C1TxReqInterrupt  _OC14Interrupt                       */
00095 /* _U2RXInterrupt     _C2TxReqInterrupt  _IC14Interrupt                       */
00096 /* _U2TXInterrupt     _QEI2Interrupt     _OC15Interrupt                       */
00097 /* _SPI2ErrInterrupt  _U3ErrInterrupt    _IC15Interrupt                       */
00098 /* _SPI2Interrupt     _U3RXInterrupt     _OC16Interrupt                       */
00099 /* _C1RxRdyInterrupt  _U3TXInterrupt     _IC16Interrupt                       */
00100 /* _C1Interrupt       _USB1Interrupt     _ICDInterrupt                        */
00101 /* _DMA3Interrupt     _U4ErrInterrupt    _PWMSpEventMatchInterrupt            */
00102 /* _IC3Interrupt      _U4RXInterrupt     _PWMSecSpEventMatchInterrupt         */
00103 /*                                                                            */
00104 /* For alternate interrupt vector naming, simply add 'Alt' between the prim.  */
00105 /* interrupt vector name '_' and the first character of the primary interrupt */
00106 /* vector name.  There is no Alternate Vector or 'AIVT' for the 33E family.   */
00107 /*                                                                            */
00108 /* For example, the vector name _ADC2Interrupt becomes _AltADC2Interrupt in   */
00109 /* the alternate vector table.                                                */
00110 /*                                                                            */
00111 /* Example Syntax:                                                            */
00112 /*                                                                            */
00113 /* void __attribute__((interrupt,auto_psv)) <Vector Name>(void)               */
00114 /* {                                                                          */
00115 /*     <Clear Interrupt Flag>                                                 */
00116 /* }                                                                          */
00117 /*                                                                            */
00118 /* For more comprehensive interrupt examples refer to the C30 (MPLAB C        */
00119 /* Compiler for PIC24 MCUs and dsPIC DSCs) User Guide in the                   */
00120 /* <C30 compiler instal directory>/doc directory for the latest compiler      */
00121 /* release.  For XC16, refer to the MPLAB XC16 C Compiler User's Guide in the */
00122 /* <XC16 compiler instal directory>/doc folder.                               */
00123 /*                                                                            */
00124
00125 /* Interrupt Routines                                                         */
00126
00128 void __attribute__((__interrupt__, no_auto_psv)) _T1Interrupt(void)
00129 {
00130     _T1IF = 0; //Clear Timer1 interrupt flag
00131     static int count=0;
00132
00133     if (count>=SENSOR_TIME-1)
00134     {
00135         count=0;
00136         i2c_status_t status;
00137         //putsUART("Hello World\n");
00138         //Anfrage Temperatur-Sensor
00139         exchangeI2C(0b1001000, 1, &write_data_buffer_temp, 2, read_data_buffer_temp, &status);
00140         //Anfrage Licht-Sensor
```

```
00141         exchangeI2C(0b0100011, 1, &write_data_buffer_light, 2, read_data_buffer_light, &status);
00142         print_sensor_values();
00143     }
00144     else
00145     {
00146         count++;
00147     }
00148
00149 }
00150
00151
00152 /* TODO Add interrupt routine code here. */
```

5.9 main.c-Dateireferenz

```
#include <xc.h>
#include <stdint.h>
#include <stdbool.h>
#include <string.h>
#include <stdio.h>
#include <stdlib.h>
#include "UART.h"
#include "I2C.h"
#include "system.h"
#include "user.h"
```

Include-Abhängigkeitsdiagramm für main.c:



Makrodefinitionen

- #define MAIN
- #define HEARTBEAT_MS 1

Funktionen

- int16_t main (void)

5.9.1 Makro-Dokumentation

5.9.1.1 HEARTBEAT_MS

```
#define HEARTBEAT_MS 1
```

Definiert in Zeile 31 der Datei main.c.

5.9.1.2 MAIN

```
#define MAIN
```

Definiert in Zeile 12 der Datei main.c.

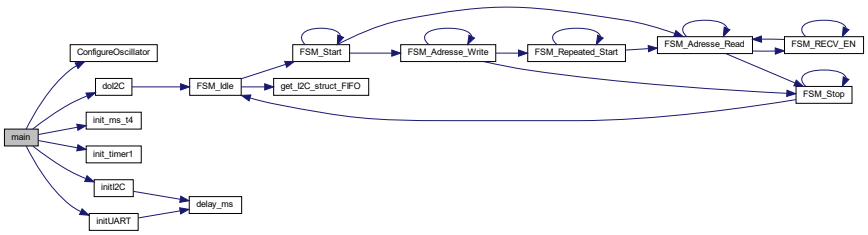
5.9.2 Dokumentation der Funktionen

5.9.2.1 main()

```
int16_t main (
            void  )
```

Definiert in Zeile 36 der Datei main.c.

Hier ist ein Graph, der zeigt, was diese Funktion aufruft:



5.10 main.c

gehe zur Dokumentation dieser Datei

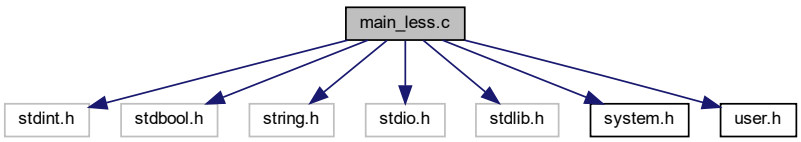
```
00001 /*TODO
00002  *  Testen
00003     Doku mit DoxyGen
00004     Lichtsensor testen
00005     FSM in Interrupt
00006  *
00007  */
00008
00009
00010 /* Files to Include                                                  */
00011
00012 #define MAIN
00013 #include <xc.h>
00014
00015 #include <stdint.h>        /* Includes uint16_t definition                  */
00016 #include <stdbool.h>       /* Includes true/false definition                */
00017 #include <string.h>
00018 #include <stdio.h>
00019 #include <stdlib.h>
00020
00021 #include "UART.h"
00022 #include "I2C.h"
00023
00024 #include "system.h"        /* System funct/params, like osc/peripheral config */
00025 #include "user.h"          /* User funct/params, such as InitApp              */
00026
00027
00028
00029 /* Global Variable Declaration                                      */
00030
00031 #define HEARTBEAT_MS 1
00032
00033 /* Main Program                                                     */
00034
00036 int16_t main(void)
00037 {
00038     DELAY_ANPASSUNG = ((SYS_FREQ/96)*2180ull)/1000000ull; //Berechnung der Delay Anpassung
00039     uint16_t Count = 0;
00040
00041     ConfigureOscillator();
00042     initUART();
00043     init_timer1();
00044     init_ms_t4();
00045     initI2C();
00046
00047     _RP66R = _RPOUT_U1TX; //UART Pin Mapping
00048     RPINR18bits.U1RXR = 0b1011000;
00049
00050
00051     write_data_buffer_temp=0b00000000;
00052     write_data_buffer_light=0b00010000;
00053     while(1)
00054     {
00055         if(_T4IF)
00056         {
00057             _T4IF=0;
00058             Count++;
00059             if (Count >= HEARTBEAT_MS)
00060             {
00061                 Count = 0;
00062                 doI2C();
00063             }
00064         }
00065     }
00066 }
```

5.11 main_less.c-Dateireferenz

```
#include <stdint.h>
#include <stdbool.h>
#include <string.h>
#include <stdio.h>
#include <stdlib.h>
#include "system.h"
```

```
#include "user.h"
```

Include-Abhängigkeitsdiagramm für main_less.c:



Datenstrukturen

- struct Buffer

Makrodefinitionen

- #define HEARTBEAT_MS 1
- #define BAUDRATE 9600
- #define BRGVAL ((FCY/BAUDRATE)/16)-1
- #define BUFFER_FAIL 0
- #define BUFFER_SUCCESS 1
- #define BUFFER_SIZE 128
- #define I2C_SCL _RA2
- #define I2C_SDA _RA3
- #define I2C_SCL_TRIS _TRISA2
- #define I2C_SDA_TRIS _TRISA3

Typdefinitionen

- typedef void ∗(∗ StateFunc) ()

Funktionen

- void init_ms_t4 (void)
- int16_t putsUART (const char ∗str)
- int16_t getcFIFO_TX (volatile uint16_t ∗c)
- int16_t putcFIFO_TX (char c)
- void ∗ FSM2_Idle (void)
- void ∗ FSM2_Start (void)
- void ∗ FSM2_Adresse (void)
- void ∗ FSM2_ACK_Receive (void)
- void ∗ FSM2_Data_Receive (void)
- void ∗ FSM2_Stop (void)
- void Temp_FSM2 (void)
- void delay_ms (uint16_t milliseconds)

  Delay in ms Blockierende Delay Funktion, eventuell nicht perfekte Verzögerung mit Hilfe einer for Schleife.
- void _T1Interrupt (void)
- void initUART ()
- void _U1TXInterrupt (void)
- int16_t putcUART (char c)
- void init_timer1 ()
- void initI2C ()

  Initialisiert die I2C-Kommunikation.
- int16_t main (void)

Variablen

- uint32_t DELAY_ANPASSUNG
- uint8_t data [2]
- Buffer FIFO = {{}, 0, 0}

5.11.1 Makro-Dokumentation

5.11.1.1 BAUDRATE

```
#define BAUDRATE 9600
```

Definiert in Zeile 32 der Datei main_less.c.

5.11.1.2 BRGVAL

```
#define BRGVAL ((FCY/BAUDRATE)/16)-1
```

Definiert in Zeile 33 der Datei main_less.c.

5.11.1.3 BUFFER_FAIL

```
#define BUFFER_FAIL 0
```

Definiert in Zeile 36 der Datei main_less.c.

5.11.1.4 BUFFER_SIZE

```
#define BUFFER_SIZE 128
```

Definiert in Zeile 38 der Datei main_less.c.

5.11.1.5 BUFFER_SUCCESS

```
#define BUFFER_SUCCESS 1
```

Definiert in Zeile 37 der Datei main_less.c.

5.11.1.6 HEARTBEAT_MS

```
#define HEARTBEAT_MS 1
```

Definiert in Zeile 29 der Datei main_less.c.

5.11.1.7 I2C_SCL

```
#define I2C_SCL _RA2
```

Definiert in Zeile 42 der Datei main_less.c.

5.11.1.8 I2C_SCL_TRIS

```
#define I2C_SCL_TRIS _TRISA2
```

Definiert in Zeile 44 der Datei main_less.c.

5.11.1.9 I2C_SDA

```
#define I2C_SDA _RA3
```

Definiert in Zeile 43 der Datei main_less.c.

5.11.1.10 I2C_SDA_TRIS

```
#define I2C_SDA_TRIS _TRISA3
```

Definiert in Zeile 45 der Datei main_less.c.

5.11.2 Dokumentation der benutzerdefinierten Typen

5.11.2.1 StateFunc

```
typedef void *(* StateFunc) ()
```

Definiert in Zeile 58 der Datei main_less.c.

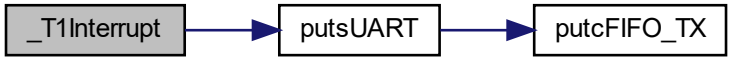
5.11.3 Dokumentation der Funktionen

5.11.3.1 _T1Interrupt()

```
void _T1Interrupt (
            void  )
```

Definiert in Zeile 91 der Datei main_less.c.

Hier ist ein Graph, der zeigt, was diese Funktion aufruft:



5.11.3.2 _U1TXInterrupt()

```
void _U1TXInterrupt (
            void  )
```

Definiert in Zeile 136 der Datei main_less.c.

Hier ist ein Graph, der zeigt, was diese Funktion aufruft:

5.11.3.3 delay_ms()

```
void delay_ms (
            uint16_t milliseconds )
```

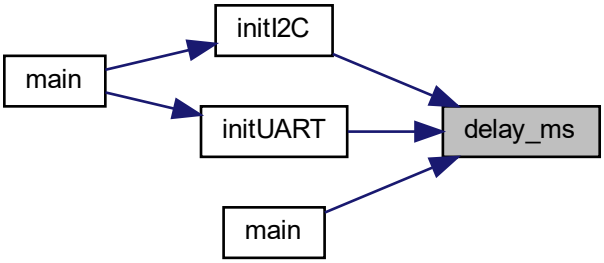
Delay in ms Blockierende Delay Funktion, eventuell nicht perfekte Verzögerung mit Hilfe einer for Schleife.

Parameter

in	*milliseconds*	Verzögerungszeit in millisekunden

Definiert in Zeile 85 der Datei main_less.c.

Hier ist ein Graph der zeigt, wo diese Funktion aufgerufen wird:

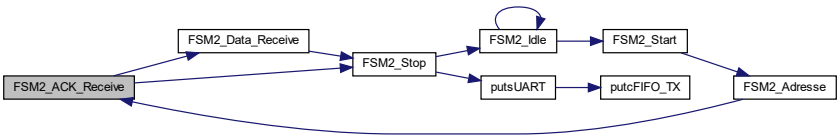


5.11.3.4 FSM2_ACK_Receive()

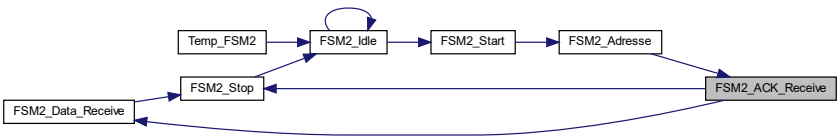
```
void * FSM2_ACK_Receive (
            void  )
```

Definiert in Zeile 321 der Datei main_less.c.

Hier ist ein Graph, der zeigt, was diese Funktion aufruft:



Hier ist ein Graph der zeigt, wo diese Funktion aufgerufen wird:

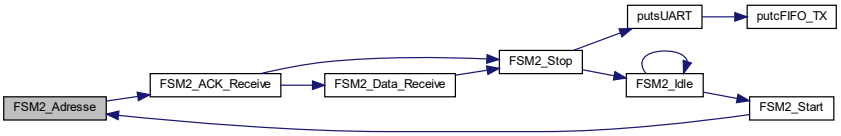


5.11.3.5 FSM2_Adresse()

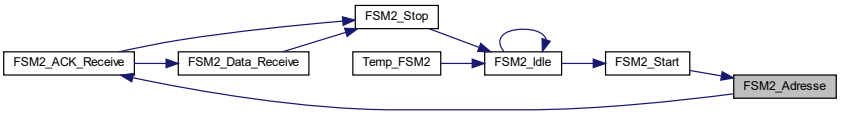
```
void * FSM2_Adresse (
            void  )
```

Definiert in Zeile 313 der Datei main_less.c.

Hier ist ein Graph, der zeigt, was diese Funktion aufruft:



Hier ist ein Graph der zeigt, wo diese Funktion aufgerufen wird:

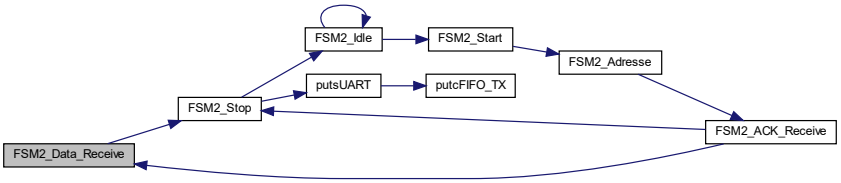


5.11.3.6 FSM2_Data_Receive()

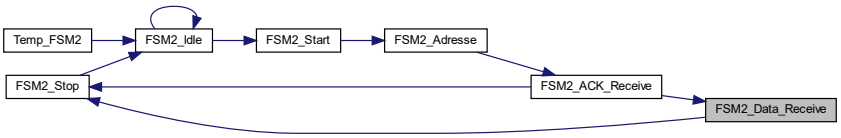
```
void * FSM2_Data_Receive (
            void  )
```

Definiert in Zeile 330 der Datei main_less.c.

Hier ist ein Graph, der zeigt, was diese Funktion aufruft:



Hier ist ein Graph der zeigt, wo diese Funktion aufgerufen wird:

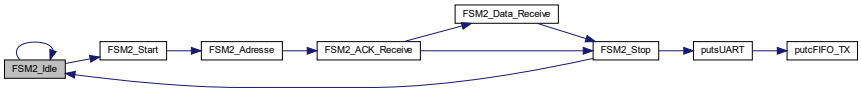


5.11.3.7 FSM2_Idle()

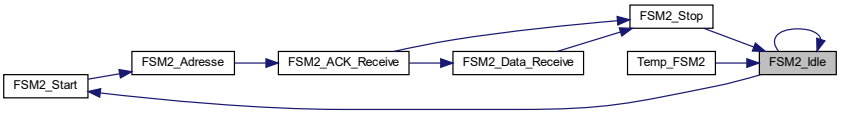
```
void * FSM2_Idle (
            void  )
```

Definiert in Zeile 294 der Datei main_less.c.

Hier ist ein Graph, der zeigt, was diese Funktion aufruft:



Hier ist ein Graph der zeigt, wo diese Funktion aufgerufen wird:

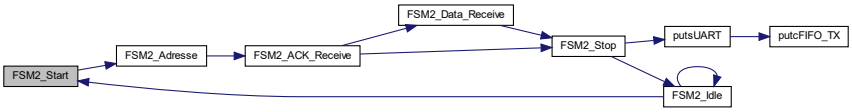


5.11.3.8 FSM2_Start()

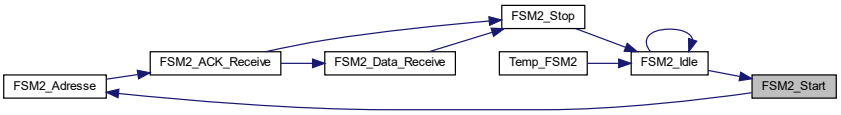
```
void * FSM2_Start (
            void  )
```

Definiert in Zeile 306 der Datei main_less.c.

Hier ist ein Graph, der zeigt, was diese Funktion aufruft:



Hier ist ein Graph der zeigt, wo diese Funktion aufgerufen wird:

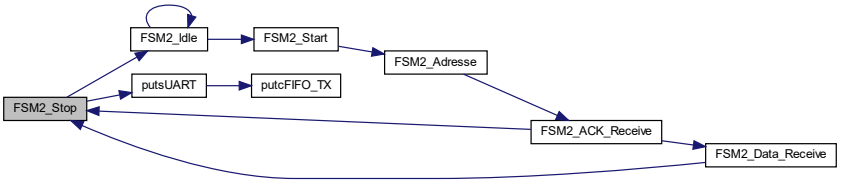


5.11.3.9 FSM2_Stop()

```
void * FSM2_Stop (
            void  )
```

Definiert in Zeile 352 der Datei main_less.c.

Hier ist ein Graph, der zeigt, was diese Funktion aufruft:



Hier ist ein Graph der zeigt, wo diese Funktion aufgerufen wird:

5.11.3.10 getcFIFO_TX()

```
int16_t getcFIFO_TX (
            volatile uint16_t * c )
```

Definiert in Zeile 165 der Datei main_less.c.

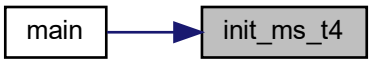
Hier ist ein Graph der zeigt, wo diese Funktion aufgerufen wird:

5.11.3.11 init_ms_t4()

```
void init_ms_t4 (
            void  )
```

Definiert in Zeile 114 der Datei system.c.

Hier ist ein Graph der zeigt, wo diese Funktion aufgerufen wird:

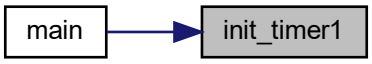


5.11.3.12 init_timer1()

```
void init_timer1 (
            void  )
```

Definiert in Zeile 210 der Datei main_less.c.

Hier ist ein Graph der zeigt, wo diese Funktion aufgerufen wird:



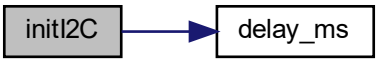
5.11.3.13 initI2C()

```
void initI2C (
             void  )
```

Initialisiert die I2C-Kommunikation.

Definiert in Zeile 234 der Datei main_less.c.

Hier ist ein Graph, der zeigt, was diese Funktion aufruft:



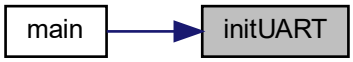
5.11.3.14 initUART()

```
void initUART (
             void  )
```

Definiert in Zeile 100 der Datei main_less.c.

Hier ist ein Graph, der zeigt, was diese Funktion aufruft:

Hier ist ein Graph der zeigt, wo diese Funktion aufgerufen wird:

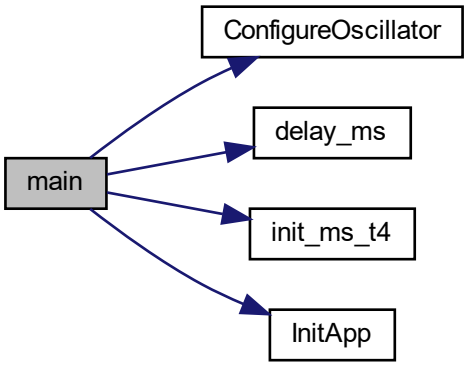


5.11.3.15 main()

```
int16_t main (
            void  )
```

Definiert in Zeile 376 der Datei main_less.c.

Hier ist ein Graph, der zeigt, was diese Funktion aufruft:

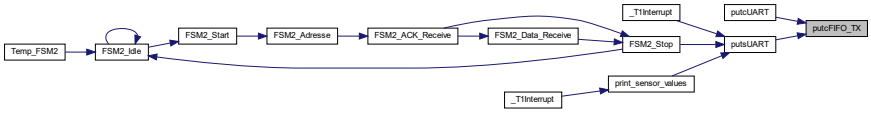


5.11.3.16 putcFIFO_TX()

```
int16_t putcFIFO_TX (
            char c )
```

Definiert in Zeile 147 der Datei main_less.c.

Hier ist ein Graph der zeigt, wo diese Funktion aufgerufen wird:



5.11.3.17 putcUART()

```
int16_t putcUART (
            char c )
```

Definiert in Zeile 180 der Datei main_less.c.

Hier ist ein Graph, der zeigt, was diese Funktion aufruft:

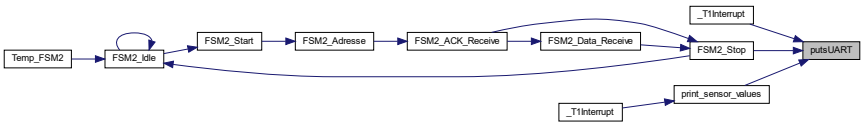
5.11.3.18 putsUART()

```
int16_t putsUART (
            const char * str )
```

Definiert in Zeile 190 der Datei main_less.c.

Hier ist ein Graph, der zeigt, was diese Funktion aufruft:

Hier ist ein Graph der zeigt, wo diese Funktion aufgerufen wird:



5.11.3.19 Temp_FSM2()

```
void Temp_FSM2 (
            void  )
```

Definiert in Zeile 227 der Datei main_less.c.

Hier ist ein Graph, der zeigt, was diese Funktion aufruft:

5.11.4 Variablen-Dokumentation

5.11.4.1 data

```
uint8_t data[2]
```

Definiert in Zeile 46 der Datei main_less.c.

5.11.4.2 DELAY_ANPASSUNG

```
uint32_t DELAY_ANPASSUNG
```

Definiert in Zeile 39 der Datei main_less.c.

5.11.4.3 FIFO

```
Buffer FIFO = {{}, 0, 0}
```

Definiert in Zeile 56 der Datei main_less.c.

5.12 main_less.c

gehe zur Dokumentation dieser Datei

```
00001
00002 /* Files to Include                                                 */
00003
00005 /* Device header file */
00006 #if defined(__XC16__)
00007     #include <xc.h>
00008 #elif defined(__C30__)
00009     #if defined(__dsPIC33E__)
00010         #include <p33Exxxx.h>
00011     #elif defined(__dsPIC33F__)
00012         #include <p33Fxxxx.h>
00013     #endif
00014 #endif
00015
00016
00017 #include <stdint.h>        /* Includes uint16_t definition                 */
00018 #include <stdbool.h>       /* Includes true/false definition               */
00019 #include <string.h>
00020 #include <stdio.h>
00021 #include <stdlib.h>
00022
00023 #include "system.h"        /* System funct/params, like osc/peripheral config */
00024 #include "user.h"          /* User funct/params, such as InitApp           */
00025
00026
00027 /* Global Variable Declaration                                         */
00028
00029 #define HEARTBEAT_MS 1
00030 //UART
00031
00032 #define BAUDRATE 9600
00033 #define BRGVAL ((FCY/BAUDRATE)/16)-1
00034 //FIFO
00035
00036 #define BUFFER_FAIL     0
00037 #define BUFFER_SUCCESS  1
00038 #define BUFFER_SIZE 128
00039 uint32_t DELAY_ANPASSUNG;
00040
00041 //I2C
00042 #define I2C_SCL      _RA2
00043 #define I2C_SDA      _RA3
00044 #define I2C_SCL_TRIS _TRISA2
00045 #define I2C_SDA_TRIS _TRISA3
00046 uint8_t data[2];
00047
00048 /*Typen-Definitionen*****************************************************/
00049
00050 typedef struct {
00051   uint8_t data[BUFFER_SIZE];
00052   uint8_t read; // zeigt auf das Feld mit dem ältesten Inhalt
00053   uint8_t write; // zeigt immer auf leeres Feld
00054 }Buffer;
00055
00056 Buffer FIFO = {{}, 0, 0};
00057
00058 typedef void *(*StateFunc)();
00059
00060
00061 /*Prototypes*************************************************************/
00062 void init_ms_t4(void);
00063
00064 int16_t putsUART(const char *str);
00065 int16_t getcFIFO_TX(volatile uint16_t *c);
00066 //int16_t getcFIFO_RX(char *c);
00067
00068 int16_t putcFIFO_TX(char c);
00069 //int16_t putcFIFO_RX(char c);
00070
```

```
00071 void *FSM2_Idle(void);
00072 void *FSM2_Start(void);
00073 void *FSM2_Adresse(void);
00074 void *FSM2_ACK_Receive(void);
00075 void *FSM2_Data_Receive(void);
00076 void *FSM2_Stop(void);
00077 void Temp_FSM2(void);
00078
00079 /*Funktionen*****************************************************************/
00085 void delay_ms(uint16_t milliseconds) {
00086     uint32_t i=0;
00087     for (i=0;i<(DELAY_ANPASSUNG*(uint32_t)milliseconds);i++){
00088          }
00089 }
00090
00091 void __attribute__((__interrupt__, no_auto_psv)) _T1Interrupt(void)
00092     {
00093         _T1IF = 0; //Clear Timer1 interrupt flag
00094
00095         putsUART("Hello World\n");
00096
00097     }
00098
00099 //UART
00100 void initUART(){
00101     U1MODEbits.STSEL = 0; // 1-Stop bit
00102     U1MODEbits.PDSEL = 0; // No Parity, 8-Data bits
00103     U1MODEbits.ABAUD = 0; // Auto-Baud disabled
00104     U1MODEbits.UEN = 0;
00105     U1MODEbits.LPBACK = 0;
00106     U1MODEbits.RXINV = 0;
00107     //U1MODEbits.ALTIO = 0;
00108
00109     U1MODEbits.URXINV = 0;
00110     U1MODEbits.RTSMD = 0;
00111
00112     U1MODEbits.BRGH = 0; // Standard-Speed mode
00113     U1BRG = BRGVAL; // Baud Rate setting for 9600
00114
00115     U1STAbits.UTXISEL0 = 0; // Interrupt after one TX character is transmitted
00116     U1STAbits.UTXISEL1 = 0;
00117     U1STAbits.UTXBRK = 0;
00118     U1STAbits.ADDEN = 0;
00119     U1STAbits.UTXINV = 0;
00120     U1STAbits.URXISEL = 0;
00121     U1STA = U1STA | 0b0001000000000000;
00122     //_URXEN = 1;
00123
00124     //_U1RXIE = 1; // Enable UART RX interrupt
00125
00126     U1MODEbits.UARTEN = 1; // Enable UART
00127     //delay_ms(2);
00128     U1STAbits.UTXEN = 1; // Enable UART TX
00129
00130     /* Wait at least 105 microseconds (1/9600) before sending first char */
00131     delay_ms(2);
00132     _U1TXIE = 1; // Enable UART TX interrupt
00133
00134 }
00135
00136 void __attribute__((__interrupt__)) _U1TXInterrupt(void)
00137 {
00138     _U1TXIF = 0; // Clear TX Interrupt flag
00139
00140     getcFIFO_TX(&U1TXREG);
00141
00142 }
00143
00144
00145
00146
00147 int16_t putcFIFO_TX(char c)
00148 {
00149   //if (buffer.write >= BUFFER_SIZE)
00150   //  buffer.write = 0; // erhöht sicherheit
00151   _LATF0 = 1;
00152   if ( ( FIFO.write + 1 == FIFO.read ) ||
00153        ( FIFO.read == 0 && FIFO.write + 1 == BUFFER_SIZE ) )
00154     return BUFFER_FAIL; // voll
00155
00156   FIFO.data[FIFO.write] = c;
00157
00158   FIFO.write++;
00159   if (FIFO.write >= BUFFER_SIZE)
00160     FIFO.write = 0;
00161
00162   return BUFFER_SUCCESS;
```

```
00163 }
00164
00165 int16_t getcFIFO_TX(volatile uint16_t *c)
00166 {
00167   _LATF0 = 1;
00168   if (FIFO.read == FIFO.write)
00169     return BUFFER_FAIL;
00170
00171   *c = FIFO.data[FIFO.read];
00172
00173   FIFO.read++;
00174   if (FIFO.read >= BUFFER_SIZE)
00175     FIFO.read = 0;
00176
00177   return BUFFER_SUCCESS;
00178 }
00179
00180 int16_t putcUART(char c){
00181     _LATF0 = 1;
00182     _GIE = 0; // Interrupts ausschalten
00183     int16_t erfolg = putcFIFO_TX(c);
00184     _GIE = 1;
00185     return erfolg;
00186
00187
00188 }
00189
00190 int16_t putsUART(const char *str) {
00191     _LATF0 = 1;
00192     uint16_t i;
00193     uint16_t length = strlen(str);
00194
00195     _GIE = 0;   //Global Interrupt disable
00196     for(i = 0; i < length; i++) {
00197         //uint16_t ret = putcFIFO_TX(str[i]);
00198         if(! putcFIFO_TX(str[i]))
00199             break;
00200     }
00201     _GIE = 1;
00202     int16_t erfolg = -i;
00203     if(erfolg == -length)
00204         erfolg *= -1;
00205     _U1TXIF = 1; //Interuppt Routine Starten um FIFO-Inhalt zu senden
00206     return erfolg;
00207 }
00208
00209 //Timer1
00210 void init_timer1(){
00211     __builtin_write_OSCCONL(0b00000011); //SOSC aktivieren
00212     T1CONbits.TON = 0; // Disable Timer
00213     T1CONbits.TCS = 1; // Select external clock
00214     T1CONbits.TSYNC = 0; // Disable Synchronization
00215     T1CONbits.TCKPS = 0b00; // Select 1:1 Prescaler
00216     TMR1 = 0x00; // Clear timer register
00217     PR1 = 32767; // Load the period value, Quarztakt
00218
00219     IPC0bits.T1IP = 2; // Set Timer 1 Interrupt Priority Level
00220     IFS0bits.T1IF = 0; // Clear Timer 1 Interrupt Flag
00221     IEC0bits.T1IE = 1; // Enable Timer1 interrupt
00222     T1CONbits.TON = 1; // Start Timer
00223 }
00224
00225 //I2C
00226
00227 void Temp_FSM2(void)
00228 {
00229      static StateFunc statefunc = FSM2_Idle;
00230
00231      statefunc = (StateFunc)(*statefunc)();
00232 }
00233
00234 void initI2C(){
00235     I2C2CONbits.A10M = 0;
00236     I2C2BRG = 245; //100kHz
00237
00238     // Einschalten I2C mit eigenem Workaround, I2C Peripheriemodul kann hier leider nicht verwendet
      werden
00239     I2C_SDA_TRIS = 1;    // Pins wie einen Open-Kollektor-Treiber verwenden, d.h. 1 - hochohmig, 0
      wird getrieben
00240     I2C_SCL_TRIS = 1;
00241     I2C_SDA = 0;
00242     I2C_SCL = 0;
00243
00244         int j;
00245         for (j=0; j<=9; j++)    // takten bis min 1 Byte
00246         {
00247           I2C_SCL_TRIS = 0; delay_ms(1); // 5 us wÃ¤ren ausreichend ...100 kBaud
```

```
00248           I2C_SCL_TRIS = 1; delay_ms(1);
00249         }
00250         // Start Condition senden
00251         I2C_SCL_TRIS = 0;  delay_ms(1);
00252         I2C_SDA_TRIS = 0;  delay_ms(1);
00253         // Stop Condition senden
00254         I2C_SCL_TRIS = 1;  delay_ms(1);
00255         I2C_SDA_TRIS = 1;  delay_ms(1);
00256
00257         // Nun I2C erst anschalten
00258         _MI2C2IF = 0; //Interrupt falls noetig
00259         _MI2C2IE = 0;
00260         I2C2CONbits.I2CEN = 1;
00261
00262     //Sensor Pointer auf TEMP Register setzten
00263     I2C2CONbits.SEN=1; //start
00264     while(I2C2CONbits.SEN==1){}
00265
00266     //Tx Device address + Write bit
00267     I2C2TRN=0b10010000;
00268     while(I2C2STATbits.TRSTAT==1){}
00269
00270     if (I2C2STATbits.ACKSTAT==1){   //if NACK received, generate stop condition and exit
00271         I2C2STATbits.ACKSTAT=0;
00272         I2C2CONbits.PEN=1;
00273         while(I2C2CONbits.PEN==1){} //wait for the stop interrupt;
00274         return;
00275     }
00276
00277     //Tx Register Address
00278     I2C2TRN=0b00000000; //Pointer auf TEMP REGISTER setzten
00279     while(I2C2STATbits.TRSTAT==1){}
00280
00281     if (I2C2STATbits.ACKSTAT==1){   //if NACK received, generate stop condition and exit
00282         I2C2STATbits.ACKSTAT=0;
00283         I2C2CONbits.PEN=1;
00284         while(I2C2CONbits.PEN==1){} //wait for the stop interrupt;
00285         return;
00286     }
00287
00288     I2C2CONbits.PEN=1; //stop
00289     while(I2C2CONbits.PEN==1){} //wait for the stop interrupt
00290 }
00291
00292
00293
00294 void *FSM2_Idle(void)
00295 {
00296     static int c = 0;
00297     if (c>=999){
00298         c=0;
00299         return FSM2_Start;
00300     }
00301     c++;
00302     return FSM2_Idle;
00303
00304 }
00305
00306 void *FSM2_Start(void)
00307 {
00308     I2C2CONbits.SEN=1; //Start
00309     while(I2C2CONbits.SEN==1){}
00310     return FSM2_Adresse;
00311 }
00312
00313 void *FSM2_Adresse(void)
00314 {
00315     //Tx Device address + Read bit
00316     I2C2TRN=0b10010001;
00317     while(I2C2STATbits.TRSTAT==1){} //Warten solange übertragen wird
00318     return FSM2_ACK_Receive;
00319 }
00320
00321 void *FSM2_ACK_Receive(void)
00322 {
00323     if (I2C2STATbits.ACKSTAT==1){   //if NACK received, generate stop condition and exit
00324         I2C2STATbits.ACKSTAT=0;
00325         return FSM2_Stop;
00326     }
00327     return FSM2_Data_Receive;
00328 }
00329
00330 void *FSM2_Data_Receive(void)
00331 {
00332     int N=2; //2 bytes empfangen
00333     int i;
00334
```

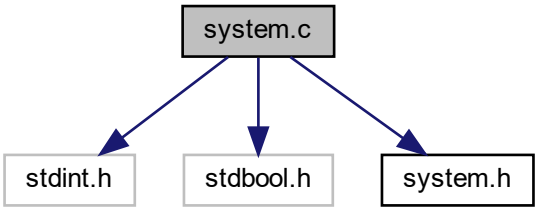
```
00335     for(i=0;i<N;i++){
00336         I2C2CONbits.RCEN=1; //Empfangen aktivieren
00337         while(I2C2CONbits.RCEN==1){} //RCEN cleared automatically when SSP1IF goes high
00338 
00339         data[i]=I2C2RCV;
00340 
00341         //ACK sequence
00342         if (i<N-1){ I2C2CONbits.ACKDT=0; } //jedes byte mit ACK bestätigen
00343         else {I2C2CONbits.ACKDT=1;} //send NACK if this is the last Byte
00344 
00345         I2C2CONbits.ACKEN=1; //start ack/nack sequence
00346         while(I2C2CONbits.ACKEN==1){}
00347 
00348     } //end for loop
00349     return FSM2_Stop;
00350 }
00351 
00352 void *FSM2_Stop(void)
00353 {
00354     I2C2CONbits.PEN=1;
00355     while(I2C2CONbits.PEN==1){} //wait for the stop interrupt
00356 
00357     float temp = data[0]«8|data[1];
00358     char str[16];
00359     sprintf(str,"%f",temp/256);
00360     putsUART("Temperatur: ");
00361     putsUART(str);
00362     putsUART("°C");
00363     putsUART("\n");
00364 
00365     return FSM2_Idle;
00366 }
00367 
00368 
00369 
00370 /* Main Program                                                              */
00371 
00374 
00376 int16_t main(void)
00377 {
00378     DELAY_ANPASSUNG = ((SYS_FREQ/96)*2180ull)/1000000ull; //Berechnung der Delay Anpassung
00379     //uint16_t Count = 0;
00380     /* Configure the oscillator for the device */
00381     ConfigureOscillator();
00382     /* Initialize IO ports and peripherals */
00383     InitApp();
00384 
00385     //initUART();
00386     //init_timer1();
00387     init_ms_t4();
00388     //initI2C();
00389 
00390 
00391     TRISBbits.TRISB8 = 0; //LED0 als Ausgang
00392     ANSELBbits.ANSB8 = 0; //LED0 als Digitaler Ausgang
00393 
00394     TRISBbits.TRISB9 = 0; //LED als Ausgang
00395     ANSELBbits.ANSB9 = 0;
00396 
00397    //Taster als Eingänge
00398     _TRISG12 = 1;
00399     //Pull-up Widerstände einschalten
00400     _CNPUG12 = 1;
00401 
00402 
00403     _RP66R = _RPOUT_U1TX; //UART Pin Mapping
00404     RPINR18bits.U1RXR = 0b1011000;
00405     /* TODO <INSERT USER APPLICATION CODE HERE> */
00406 
00407     while(1)
00408     {
00409         PORTBbits.RB8=1;
00410         delay_ms(200);
00411         PORTBbits.RB8=0;
00412         delay_ms(200);
00413         //if(_T4IF)
00414         //{
00415             //_T4IF=0;
00416             //Count++;
00417             //if (Count >= HEARTBEAT_MS)
00418             //{
00419                 //Count = 0;
00420                 //Temp_FSM2();
00421             //}
00422 
00423         //}
00424     }
```

```
00425 }
```

5.13 system.c-Dateireferenz

```
#include <stdint.h>
#include <stdbool.h>
#include "system.h"
```

Include-Abhängigkeitsdiagramm für system.c:



Funktionen

- void ConfigureOscillator (void)
- void init_timer1 ()
- void init_ms_t4 ()
- void delay_ms (uint16_t milliseconds)

  Delay in ms Blockierende Delay Funktion, eventuell nicht perfekte Verzögerung mit Hilfe einer for Schleife.

5.13.1 Dokumentation der Funktionen

5.13.1.1 ConfigureOscillator()

```
void ConfigureOscillator (
            void  )
```

Definiert in Zeile 40 der Datei system.c.

Hier ist ein Graph der zeigt, wo diese Funktion aufgerufen wird:

5.13.1.2 delay_ms()

```
void delay_ms (
            uint16_t milliseconds )
```

Delay in ms Blockierende Delay Funktion, eventuell nicht perfekte Verzögerung mit Hilfe einer for Schleife.

Parameter

in	*milliseconds*	Verzögerungszeit in millisekunden

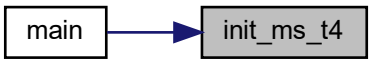
Definiert in Zeile 129 der Datei system.c.

5.13.1.3 init_ms_t4()

```
void init_ms_t4 (
            void  )
```

Definiert in Zeile 114 der Datei system.c.

Hier ist ein Graph der zeigt, wo diese Funktion aufgerufen wird:



5.13.1.4 init_timer1()

```
void init_timer1 (
            void  )
```

Definiert in Zeile 98 der Datei system.c.

5.14 system.c

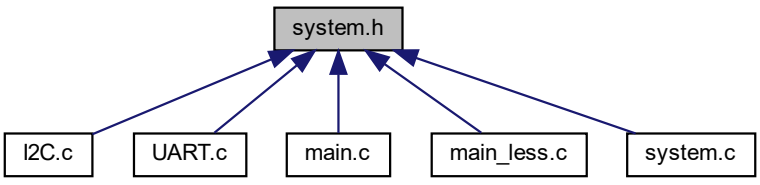
gehe zur Dokumentation dieser Datei

```
00001
00002 /* Files to Include                                                 */
00003
00005 /* Device header file */
00006 #if defined(__XC16__)
00007     #include <xc.h>
00008 #elif defined(__C30__)
00009     #if defined(__dsPIC33E__)
00010         #include <p33Exxxx.h>
00011     #elif defined(__dsPIC33F__)
00012         #include <p33Fxxxx.h>
00013     #endif
00014 #endif
00015
00016
00017 #include <stdint.h>          /* For uint16_t definition                */
00018 #include <stdbool.h>         /* For true/false definition              */
00019
00020 #include "system.h"          /* variables/params used by system.c      */
00021
00022
00023 /* System Level Functions                                             */
00024 /*                                                                    */
00025 /* Custom oscillator configuration funtions, reset source evaluation  */
00026 /* functions, and other non-peripheral microcontroller initialization */
00027 /* functions get placed in system.c.                                  */
00028 /*                                                                    */
00029
00031 /* Refer to the device Family Reference Manual Oscillator section for
00032 information about available oscillator configurations.  Typically
00033 this would involve configuring the oscillator tuning register or clock
00034 switching useing the compiler's __builtin_write_OSCCON functions.
00035 Refer to the C Compiler for PIC24 MCUs and dsPIC DSCs User Guide in the
00036 compiler installation directory /doc folder for documentation on the
00037 __builtin functions.*/
00038
00039 /* TODO Add clock switching code if appropriate.  An example stub is below.   */
00040 void ConfigureOscillator(void)
00041 {
00042     if (SYS_FREQ>7370000L) //Nur umschalten auf Primary (8 MHz) wenn höhere Frequenz erwünscht
00043     {
00044         switch (SYS_FREQ)
00045         {
00046             case 8000000L:
00047                 //PLL muss nicht konfiguriert werden
00048                 // externer Quartz mit 8Mhz
00049                 break;
00050             case 50000000L:
00051                 CLKDIVbits.PLLPOST=2; //N2=4
00052                 PLLFBD=48; //M=50
00053                 CLKDIVbits.PLLPRE=0; //N1=2
00054                 break;
00055             case 70000000L:
00056                 CLKDIVbits.PLLPOST=2; //N2=4
00057                 PLLFBD=188; //M=190
00058                 CLKDIVbits.PLLPRE=3; //N1=5
00059                 break;
00060             case 100000000L:
00061                 CLKDIVbits.PLLPOST=0; //N2=2
00062                 PLLFBD=123; //M=125
00063                 CLKDIVbits.PLLPRE=3; //N1=5
00064                 break;
00065             case 140000000L:
00066                 CLKDIVbits.PLLPOST=0; //N2=2
00067                 PLLFBD=173; //M=175
00068                 CLKDIVbits.PLLPRE=3; //N1=5
00069                 break;
00070             //default:
00071                 //#error Tets
00072         }
00073         OSCTUN = 0;
00074
00075         if (SYS_FREQ == 8000000L)
00076         {
00077             __builtin_write_OSCCONH(0x02); //Switch auf Primary ohne PLL
00078
00079             __builtin_write_OSCCONL(OSCCON | 0x01);
00080             while (OSCCONbits.COSC!= 0x02); //Warten bis gewechselt wurde
00081         }
00082
00083         else
00084         {
```

```
00085             __builtin_write_OSCCONH(0x03); //Switch auf Primary mit PLL
00086
00087             __builtin_write_OSCCONL(OSCCON | 0x01);
00088
00089             while (OSCCONbits.COSC!= 0x3); //Warten bis gewechselt wurde
00090             while (OSCCONbits.LOCK!= 1);
00091         }
00092
00093     }
00094 }
00095
00096
00097 //Timer1
00098 void init_timer1() //generiert in 1s Rythmus Interrupts
00099 {
00100     __builtin_write_OSCCONL(0b00000011); //SOSC aktivieren
00101     T1CONbits.TON = 0; // Disable Timer
00102     T1CONbits.TCS = 1; // Select external clock
00103     T1CONbits.TSYNC = 0; // Disable Synchronization
00104     T1CONbits.TCKPS = 0b00; // Select 1:1 Prescaler
00105     TMR1 = 0x00; // Clear timer register
00106     PR1 = 32767; // Load the period value, Quarztakt
00107
00108     IPC0bits.T1IP = 2; // Set Timer 1 Interrupt Priority Level
00109     IFS0bits.T1IF = 0; // Clear Timer 1 Interrupt Flag
00110     IEC0bits.T1IE = 1; // Enable Timer1 interrupt
00111     T1CONbits.TON = 1; // Start Timer
00112 }
00113
00114 void init_ms_t4() //Interrupt Flag wird jede ms gesetzt
00115 {
00116     T4CONbits.TON = 0; // Stop any 16/32-bit Timer3 operation
00117     T4CONbits.TCS = 0; // Select internal instruction cycle clock
00118
00119     T4CONbits.TGATE = 0; // Disable Gated Timer mode
00120     T4CONbits.TCKPS = 0b10; // Select 1:64 Prescaler
00121     TMR4 = 0x00; // Clear
00122     PR4 = (FCY/64000)-1; // Load 32-bit period value (lsw)
00123     //IFS0bits.T2IF = 0; // Clear Timer2 Interrupt Flag
00124     //IEC0bits.T2IE = 0; // Disable Timer2 interrupt
00125     T4CONbits.TON = 1; // Start 32-bit Timer
00126 }
00127
00128
00129 void delay_ms(uint16_t milliseconds)
00130 {
00131     uint32_t i=0;
00132     for (i=0;i<(DELAY_ANPASSUNG*(uint32_t)milliseconds);i++)
00133     {
00134     }
00135 }
00136
```

5.15 system.h-Dateireferenz

Dieser Graph zeigt, welche Datei direkt oder indirekt diese Datei enthält:



Makrodefinitionen

- #define SYS_FREQ 50000000L
- #define FCY SYS_FREQ/2

Funktionen

- void ConfigureOscillator (void)
- void delay_ms (uint16_t milliseconds)

  Delay in ms Blockierende Delay Funktion, eventuell nicht perfekte Verzögerung mit Hilfe einer for Schleife.
- void init_timer1 (void)
- void init_ms_t4 (void)

Variablen

- uint32_t DELAY_ANPASSUNG

5.15.1 Makro-Dokumentation

5.15.1.1 FCY

```
#define FCY SYS_FREQ/2
```

Definiert in Zeile 15 der Datei system.h.

5.15.1.2 SYS_FREQ

```
#define SYS_FREQ 50000000L
```

Definiert in Zeile 10 der Datei system.h.

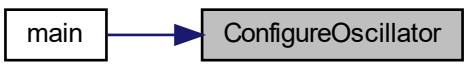
5.15.2 Dokumentation der Funktionen

5.15.2.1 ConfigureOscillator()

```
void ConfigureOscillator (
            void  )
```

Definiert in Zeile 40 der Datei system.c.

Hier ist ein Graph der zeigt, wo diese Funktion aufgerufen wird:



5.15.2.2 delay_ms()

```
void delay_ms (
            uint16_t milliseconds )
```

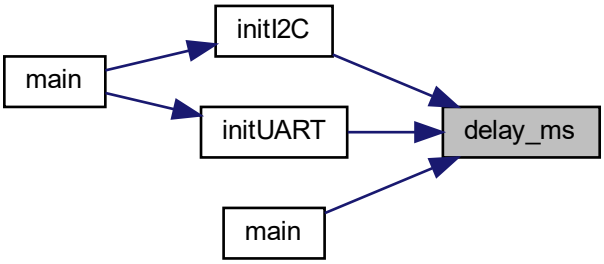
Delay in ms Blockierende Delay Funktion, eventuell nicht perfekte Verzögerung mit Hilfe einer for Schleife.

Parameter

in	*milliseconds*	Verzögerungszeit in millisekunden

Definiert in Zeile 85 der Datei main_less.c.

Hier ist ein Graph der zeigt, wo diese Funktion aufgerufen wird:



5.15.2.3 init_ms_t4()

```
void init_ms_t4 (
            void  )
```

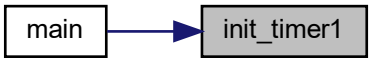
Definiert in Zeile 114 der Datei system.c.

5.15.2.4 init_timer1()

```
void init_timer1 (
            void  )
```

Definiert in Zeile 210 der Datei main_less.c.

Hier ist ein Graph der zeigt, wo diese Funktion aufgerufen wird:



5.15.3 Variablen-Dokumentation

5.15.3.1 DELAY_ANPASSUNG

```
uint32_t DELAY_ANPASSUNG  [extern]
```

Definiert in Zeile 39 der Datei main_less.c.

5.16 system.h

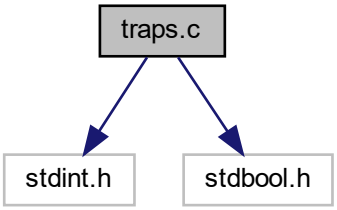
gehe zur Dokumentation dieser Datei

```
00001
00002 /* System Level #define Macros                                          */
00003
00005 /* TODO Define system operating frequency */
00006
00007 /* Microcontroller MIPs (FCY) */
00008 //#define SYS_FREQ        7370000L
00009 //#define SYS_FREQ        8000000L
00010 #define SYS_FREQ        50000000L
00011 //#define SYS_FREQ        70000000L
00012 //#define SYS_FREQ        100000000L
00013 //#define SYS_FREQ        140000000L
00014
00015 #define FCY             SYS_FREQ/2
00016
00017
00018 #ifdef MAIN
00019 uint32_t DELAY_ANPASSUNG;
00020 #else
00021 extern uint32_t DELAY_ANPASSUNG;
00022 #endif
00023
00024
00025
00026 /* System Function Prototypes                                          */
00027
00029 /* Custom oscillator configuration funtions, reset source evaluation
00030 functions, and other non-peripheral microcontroller initialization functions
00031 go here. */
00032
00033
00034 //System Prototypen
00035 void ConfigureOscillator(void); /* Handles clock switching/osc initialization */
00036 void delay_ms(uint16_t milliseconds);
00037
00038 void init_timer1(void);
00039 void init_ms_t4(void);
00040
```

5.17 traps.c-Dateireferenz

```
#include <stdint.h>
#include <stdbool.h>
```

Include-Abhängigkeitsdiagramm für traps.c:



Funktionen

- void _OscillatorFail (void)
- void _AddressError (void)
- void _StackError (void)
- void _MathError (void)
- void _DefaultInterrupt (void)

5.17.1 Dokumentation der Funktionen

5.17.1.1 _AddressError()

```
void _AddressError (
            void  )
```

Definiert in Zeile 82 der Datei traps.c.

5.17.1.2 _DefaultInterrupt()

```
void _DefaultInterrupt (
            void  )
```

Definiert in Zeile 154 der Datei traps.c.

5.17.1.3 _MathError()

```
void _MathError (
              void  )
```

Definiert in Zeile 93 der Datei traps.c.

5.17.1.4 _OscillatorFail()

```
void _OscillatorFail (
              void  )
```

Definiert in Zeile 76 der Datei traps.c.

5.17.1.5 _StackError()

```
void _StackError (
              void  )
```

Definiert in Zeile 87 der Datei traps.c.

5.18 traps.c

gehe zur Dokumentation dieser Datei

```
00001
00002 /* Files to Include                                                 */
00003
00005 /* Device header file */
00006 #if defined(__XC16__)
00007     #include <xc.h>
00008 #elif defined(__C30__)
00009     #if defined(__dsPIC33E__)
00010         #include <p33Exxxx.h>
00011     #elif defined(__dsPIC33F__)
00012         #include <p33Fxxxx.h>
00013     #endif
00014 #endif
00015
00016 #include <stdint.h>        /* Includes uint16_t definition */
00017 #include <stdbool.h>       /* Includes true/false definition */
00018
00019
00020 /* Trap Function Prototypes                                         */
00021
00023 /* <Other function prototypes for debugging trap code may be inserted here>   */
00024
00025 /* Use if INTCON2 ALTIVT=1 */
00026 void __attribute__((interrupt,no_auto_psv)) _OscillatorFail(void);
00027 void __attribute__((interrupt,no_auto_psv)) _AddressError(void);
00028 void __attribute__((interrupt,no_auto_psv)) _StackError(void);
00029 void __attribute__((interrupt,no_auto_psv)) _MathError(void);
00030
00031 #if defined(__HAS_DMA__)
00032
00033 void __attribute__((interrupt,no_auto_psv)) _DMACError(void);
00034
00035 #endif
00036
00037 #if defined(__dsPIC33F__)
00038
```

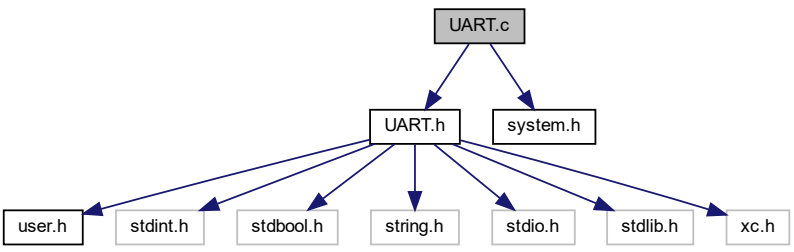
```
00039 /* Use if INTCON2 ALTIVT=0 */
00040 void __attribute__((interrupt,no_auto_psv)) _AltOscillatorFail(void);
00041 void __attribute__((interrupt,no_auto_psv)) _AltAddressError(void);
00042 void __attribute__((interrupt,no_auto_psv)) _AltStackError(void);
00043 void __attribute__((interrupt,no_auto_psv)) _AltMathError(void);
00044
00045     #if defined(__HAS_DMA__)
00046
00047     void __attribute__((interrupt,no_auto_psv)) _AltDMACError(void);
00048
00049     #endif
00050
00051 #endif
00052
00053 /* Default interrupt handler */
00054 void __attribute__((interrupt,no_auto_psv)) _DefaultInterrupt(void);
00055
00056 #if defined(__dsPIC33E__)
00057
00058 /* These are additional traps in the 33E family.  Refer to the PIC33E
00059 migration guide.  There are no Alternate Vectors in the 33E family. */
00060 void __attribute__((interrupt,no_auto_psv)) _HardTrapError(void);
00061 void __attribute__((interrupt,no_auto_psv)) _SoftTrapError(void);
00062
00063 #endif
00064
00065
00066 /* Trap Handling                                                         */
00067 /*                                                                       */
00068 /* These trap routines simply ensure that the device continuously loops  */
00069 /* within each routine.  Users who actually experience one of these traps */
00070 /* can add code to handle the error.  Some basic examples for trap code,  */
00071 /* including assembly routines that process trap sources, are available at */
00072 /* www.microchip.com/codeexamples                                        */
00073
00075 /* Primary (non-alternate) address error trap function declarations */
00076 void __attribute__((interrupt,no_auto_psv)) _OscillatorFail(void)
00077 {
00078         INTCON1bits.OSCFAIL = 0;        /* Clear the trap flag */
00079         while(1);
00080 }
00081
00082 void __attribute__((interrupt,no_auto_psv)) _AddressError(void)
00083 {
00084         INTCON1bits.ADDRERR = 0;        /* Clear the trap flag */
00085         while (1);
00086 }
00087 void __attribute__((interrupt,no_auto_psv)) _StackError(void)
00088 {
00089         INTCON1bits.STKERR = 0;         /* Clear the trap flag */
00090         while (1);
00091 }
00092
00093 void __attribute__((interrupt,no_auto_psv)) _MathError(void)
00094 {
00095         INTCON1bits.MATHERR = 0;        /* Clear the trap flag */
00096         while (1);
00097 }
00098
00099 #if defined(__HAS_DMA__)
00100
00101 void __attribute__((interrupt,no_auto_psv)) _DMACError(void)
00102 {
00103         INTCON1bits.DMACERR = 0;        /* Clear the trap flag */
00104         while (1);
00105 }
00106
00107 #endif
00108
00109 #if defined(__dsPIC33F__)
00110
00111 /* Alternate address error trap function declarations */
00112 void __attribute__((interrupt,no_auto_psv)) _AltOscillatorFail(void)
00113 {
00114         INTCON1bits.OSCFAIL = 0;        /* Clear the trap flag */
00115         while (1);
00116 }
00117
00118 void __attribute__((interrupt,no_auto_psv)) _AltAddressError(void)
00119 {
00120         INTCON1bits.ADDRERR = 0;        /* Clear the trap flag */
00121         while (1);
00122 }
00123
00124 void __attribute__((interrupt,no_auto_psv)) _AltStackError(void)
00125 {
00126         INTCON1bits.STKERR = 0;         /* Clear the trap flag */
```

```
00127         while (1);
00128 }
00129
00130 void __attribute__((interrupt,no_auto_psv)) _AltMathError(void)
00131 {
00132         INTCON1bits.MATHERR = 0;        /* Clear the trap flag */
00133         while (1);
00134 }
00135
00136     #if defined(__HAS_DMA__)
00137
00138     void __attribute__((interrupt,no_auto_psv)) _AltDMACError(void)
00139     {
00140          INTCON1bits.DMACERR = 0;        /* Clear the trap flag */
00141          while (1);
00142     }
00143
00144     #endif
00145
00146 #endif
00147
00148
00149 /* Default Interrupt Handler                                          */
00150 /*                                                                    */
00151 /* This executes when an interrupt occurs for an interrupt source with an     */
00152 /* improperly defined or undefined interrupt handling routine.                */
00153
00154 void __attribute__((interrupt,no_auto_psv)) _DefaultInterrupt(void)
00155 {
00156         while(1);
00157 }
00158
00159 #if defined(__dsPIC33E__)
00160
00161 /* These traps are new to the dsPIC33E family.  Refer to the device Interrupt
00162 chapter of the FRM to understand trap priority. */
00163 void __attribute__((interrupt,no_auto_psv)) _HardTrapError(void)
00164 {
00165     while(1);
00166 }
00167 void __attribute__((interrupt,no_auto_psv)) _SoftTrapError(void)
00168 {
00169     while(1);
00170 }
00171
00172 #endif
```

5.19 UART.c-Dateireferenz

```
#include "UART.h"
#include "system.h"
```

Include-Abhängigkeitsdiagramm für UART.c:



Funktionen

- void initUART ()

- void _U1TXInterrupt (void)
- int16_t putcFIFO_TX (char c)
- int16_t getcFIFO_TX (volatile uint16_t ∗c)
- int16_t putcUART (char c)
- int16_t putsUART (const char ∗str)

5.19.1 Dokumentation der Funktionen

5.19.1.1 _U1TXInterrupt()

```
void _U1TXInterrupt (
             void  )
```

Definiert in Zeile 42 der Datei UART.c.

Hier ist ein Graph, der zeigt, was diese Funktion aufruft:

5.19.1.2 getcFIFO_TX()

```
int16_t getcFIFO_TX (
             volatile uint16_t * c )
```

Definiert in Zeile 70 der Datei UART.c.

Hier ist ein Graph der zeigt, wo diese Funktion aufgerufen wird:

5.19.1.3 initUART()

```
void initUART (
             void  )
```

Definiert in Zeile 4 der Datei UART.c.

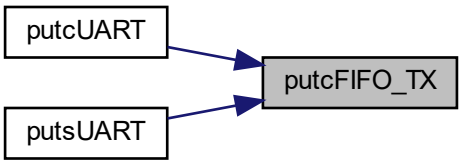
Hier ist ein Graph, der zeigt, was diese Funktion aufruft:

5.19.1.4 putcFIFO_TX()

```
int16_t putcFIFO_TX (
             char c )
```

Definiert in Zeile 49 der Datei UART.c.

Hier ist ein Graph der zeigt, wo diese Funktion aufgerufen wird:



5.19.1.5 putcUART()

```
int16_t putcUART (
            char c )
```

Definiert in Zeile 87 der Datei UART.c.

Hier ist ein Graph, der zeigt, was diese Funktion aufruft:

5.19.1.6 putsUART()

```
int16_t putsUART (
            const char * str )
```

Definiert in Zeile 96 der Datei UART.c.

Hier ist ein Graph, der zeigt, was diese Funktion aufruft:

5.20 UART.c

gehe zur Dokumentation dieser Datei

```
00001 #include "UART.h"
00002 #include "system.h"
00003
00004 void initUART()
00005 {
00006     U1MODEbits.STSEL = 0; // 1-Stop bit
00007     U1MODEbits.PDSEL = 0; // No Parity, 8-Data bits
00008     U1MODEbits.ABAUD = 0; // Auto-Baud disabled
00009     U1MODEbits.UEN = 0;
00010     U1MODEbits.LPBACK = 0;
00011     U1MODEbits.RXINV = 0;
00012     //U1MODEbits.ALTIO = 0;
00013
```

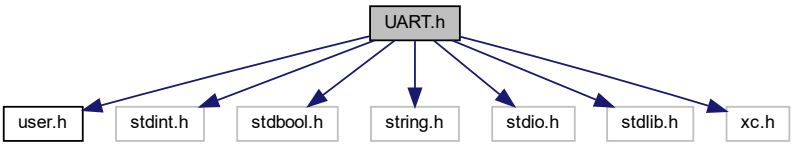
```
00014     U1MODEbits.URXINV = 0;
00015     U1MODEbits.RTSMD = 0;
00016
00017     U1MODEbits.BRGH = 0; // Standard-Speed mode
00018     U1BRG = BRGVAL; // Baud Rate setting for 9600
00019
00020     U1STAbits.UTXISEL0 = 0; // Interrupt after one TX character is transmitted
00021     U1STAbits.UTXISEL1 = 0;
00022     U1STAbits.UTXBRK = 0;
00023     U1STAbits.ADDEN = 0;
00024     U1STAbits.UTXINV = 0;
00025     U1STAbits.URXISEL = 0;
00026     U1STA = U1STA | 0b0001000000000000;
00027     //_URXEN = 1;
00028
00029     //_U1RXIE = 1; // Enable UART RX interrupt
00030
00031     U1MODEbits.UARTEN = 1; // Enable UART
00032     //delay_ms(2);
00033     U1STAbits.UTXEN = 1; // Enable UART TX
00034
00035     /* Wait at least 105 microseconds (1/9600) before sending first char */
00036     delay_ms(2);
00037     _U1TXIE = 1; // Enable UART TX interrupt
00038
00039 }
00040
00041
00042 void __attribute__((__interrupt__, no_auto_psv)) _U1TXInterrupt(void)
00043 {
00044     _U1TXIF = 0; // Clear TX Interrupt flag
00045     getcFIFO_TX(&U1TXREG);
00046
00047 }
00048
00049 int16_t putcFIFO_TX(char c)
00050 {
00051   //if (buffer.write >= BUFFER_SIZE)
00052   //  buffer.write = 0; // erhöht sicherheit
00053   _LATF0 = 1;
00054   if ( ( FIFO.write + 1 == FIFO.read ) ||
00055        ( FIFO.read == 0 && FIFO.write + 1 == BUFFER_SIZE ) )
00056   {
00057     return BUFFER_FAIL; // voll
00058   }
00059
00060   FIFO.data[FIFO.write] = c;
00061
00062   FIFO.write++;
00063   if (FIFO.write >= BUFFER_SIZE)
00064   {
00065     FIFO.write = 0;
00066   }
00067   return BUFFER_SUCCESS;
00068 }
00069
00070 int16_t getcFIFO_TX(volatile uint16_t *c)
00071 {
00072   _LATF0 = 1;
00073   if (FIFO.read == FIFO.write)
00074   {
00075     return BUFFER_FAIL;
00076   }
00077   *c = FIFO.data[FIFO.read];
00078
00079   FIFO.read++;
00080   if (FIFO.read >= BUFFER_SIZE)
00081   {
00082     FIFO.read = 0;
00083   }
00084   return BUFFER_SUCCESS;
00085 }
00086
00087 int16_t putcUART(char c)
00088 {
00089     _LATF0 = 1;
00090     _GIE = 0; // Interrupts ausschalten
00091     int16_t erfolg = putcFIFO_TX(c);
00092     _GIE = 1;
00093     return erfolg;
00094 }
00095
00096 int16_t putsUART(const char *str)
00097 {
00098     _LATF0 = 1;
00099     uint16_t i;
00100     uint16_t length = strlen(str);
```

```
00101
00102     _GIE = 0;   //Global Interrupt disable
00103     for(i = 0; i < length; i++)
00104     {
00105         //uint16_t ret = putcFIFO_TX(str[i]);
00106         if(! putcFIFO_TX(str[i]))
00107             break;
00108     }
00109     _GIE = 1;
00110     int16_t erfolg = -i;
00111     if(erfolg == -length)
00112         erfolg *= -1;
00113     _U1TXIF = 1; //Interuppt Routine Starten um FIFO-Inhalt zu senden
00114     return erfolg;
00115 }
```

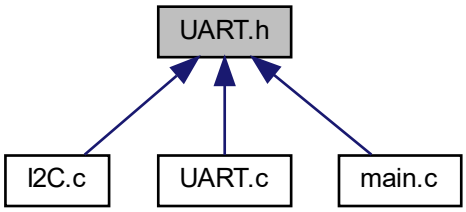
5.21 UART.h-Dateireferenz

```
#include "user.h"
#include <stdint.h>
#include <stdbool.h>
#include <string.h>
#include <stdio.h>
#include <stdlib.h>
#include <xc.h>
```

Include-Abhängigkeitsdiagramm für UART.h:



Dieser Graph zeigt, welche Datei direkt oder indirekt diese Datei enthält:



Datenstrukturen

- struct Buffer

Makrodefinitionen

- #define BAUDRATE 9600
- #define BRGVAL ((FCY/BAUDRATE)/16)-1

Funktionen

- void initUART (void)
- int16_t putsUART (const char *str)
- int16_t getcFIFO_TX (volatile uint16_t *c)
- int16_t putcFIFO_TX (char c)

Variablen

- Buffer FIFO

5.21.1 Makro-Dokumentation

5.21.1.1 BAUDRATE

```
#define BAUDRATE 9600
```

Definiert in Zeile 11 der Datei UART.h.

5.21.1.2 BRGVAL

```
#define BRGVAL ((FCY/BAUDRATE)/16)-1
```

Definiert in Zeile 12 der Datei UART.h.

5.21.2 Dokumentation der Funktionen

5.21.2.1 getcFIFO_TX()

```
int16_t getcFIFO_TX (
            volatile uint16_t * c )
```

Definiert in Zeile 165 der Datei main_less.c.

Hier ist ein Graph der zeigt, wo diese Funktion aufgerufen wird:

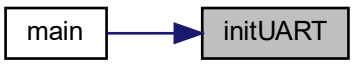
5.21.2.2 initUART()

```
void initUART (
            void  )
```

Definiert in Zeile 100 der Datei main_less.c.

Hier ist ein Graph, der zeigt, was diese Funktion aufruft:

Hier ist ein Graph der zeigt, wo diese Funktion aufgerufen wird:

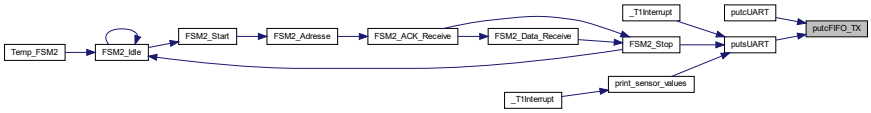


5.21.2.3 putcFIFO_TX()

```
int16_t putcFIFO_TX (
            char c )
```

Definiert in Zeile 147 der Datei main_less.c.

Hier ist ein Graph der zeigt, wo diese Funktion aufgerufen wird:



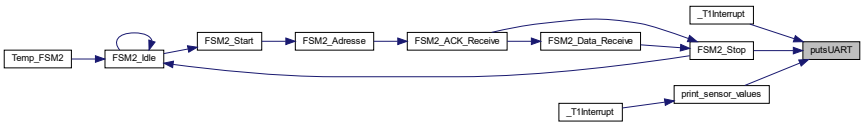
5.21.2.4 putsUART()

```
int16_t putsUART (
            const char * str )
```

Definiert in Zeile 190 der Datei main_less.c.

Hier ist ein Graph, der zeigt, was diese Funktion aufruft:

Hier ist ein Graph der zeigt, wo diese Funktion aufgerufen wird:



5.21.3 Variablen-Dokumentation

5.21.3.1 FIFO

```
Buffer FIFO  [extern]
```

Definiert in Zeile 56 der Datei main_less.c.

5.22 UART.h

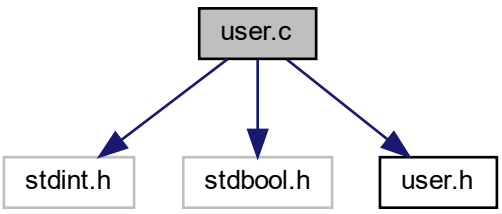
gehe zur Dokumentation dieser Datei

```
00001 #include "user.h"
00002 #include <stdint.h>        /* Includes uint16_t definition                    */
00003 #include <stdbool.h>       /* Includes true/false definition                  */
00004 #include <string.h>
00005 #include <stdio.h>
00006 #include <stdlib.h>
00007
00008 #include <xc.h>
00009
00010 //Konstanten
00011 #define BAUDRATE 9600
00012 #define BRGVAL ((FCY/BAUDRATE)/16)-1
00013
00014 //Typedefs
00015 typedef struct
00016 {
00017   uint8_t data[BUFFER_SIZE];
00018   uint8_t read; // zeigt auf das Feld mit dem ältesten Inhalt
00019   uint8_t write; // zeigt immer auf leeres Feld
00020 }Buffer;
00021
00022 #ifdef MAIN
00023 //Globale Variablen
00024 Buffer FIFO = {{}, 0, 0}; //FIFO zum Versenden über UART
00025 #else
00026 extern Buffer FIFO;
00027 #endif
00028
00029 //Prototypen
00030 void initUART(void);
00031
00032 int16_t putsUART(const char *str);
00033 int16_t getcFIFO_TX(volatile uint16_t *c);
00034 //int16_t getcFIFO_RX(char *c);
00035
00036 int16_t putcFIFO_TX(char c);
00037 //int16_t putcFIFO_RX(char c);
```

5.23 user.c-Dateireferenz

```
#include <stdint.h>
#include <stdbool.h>
#include "user.h"
```

Include-Abhängigkeitsdiagramm für user.c:



Funktionen

- void InitApp (void)
- void setLED (uint16_t nr)

5.23.1 Dokumentation der Funktionen

5.23.1.1 InitApp()

```
void InitApp (
             void  )
```

Definiert in Zeile 26 der Datei user.c.

Hier ist ein Graph der zeigt, wo diese Funktion aufgerufen wird:

5.23.1.2 setLED()

```
void setLED (
             uint16_t nr )
```

Definiert in Zeile 35 der Datei user.c.

5.24 user.c

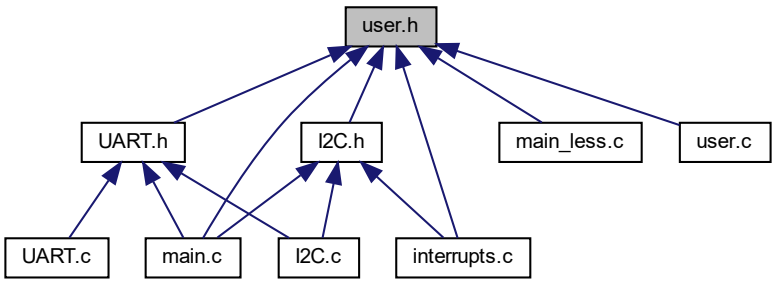
gehe zur Dokumentation dieser Datei

```
00001
00002 /* Files to Include                                                */
00003
00005 /* Device header file */
00006 #if defined(__XC16__)
00007     #include <xc.h>
00008 #elif defined(__C30__)
00009     #if defined(__dsPIC33E__)
00010         #include <p33Exxxx.h>
00011     #elif defined(__dsPIC33F__)
00012         #include <p33Fxxxx.h>
00013     #endif
00014 #endif
00015
```

```
00016 #include <stdint.h>          /* For uint16_t definition                    */
00017 #include <stdbool.h>         /* For true/false definition                  */
00018 #include "user.h"            /* variables/params used by user.c            */
00019
00020
00021 /* User Functions                                                          */
00022
00024 /* <Initialize variables in user.h and insert code for user algorithms.> */
00025
00026 void InitApp(void)
00027 {
00028     /* TODO Initialize User Ports/Peripherals/Project here */
00029
00030     /* Setup analog functionality and port direction */
00031
00032     /* Initialize peripherals */
00033 }
00034
00035 void setLED(uint16_t nr)
00036 {
00037    if (nr>=4) return;
00038    LATB = LATB | (1 « (nr+8));
00039 }
00040
00041
00042
00043 //4-bit Wort -> RB8-11
00044
00045 //uint16_t leds=0b 0000 0000 0000 1101;
00046
00047 //LATB = (LATB & ~0b0000111100000000) | ((leds«8) &0b0000111100000000);
```

5.25 user.h-Dateireferenz

Dieser Graph zeigt, welche Datei direkt oder indirekt diese Datei enthält:



Makrodefinitionen

- #define LED0 _LATB8
- #define LED1 _LATB9
- #define LED2 _LATB10
- #define LED3 _LATB11
- #define T0 !_RG12
- #define T1 !_RG13
- #define T2 !_RG14
- #define T3 !_RG15
- #define BUFFER_FAIL 0
- #define BUFFER_SUCCESS 1
- #define BUFFER_SIZE 256
- #define SENSOR_TIME 2

Funktionen

- void InitApp (void)

5.25.1 Makro-Dokumentation

5.25.1.1 BUFFER_FAIL

```
#define BUFFER_FAIL 0
```

Definiert in Zeile 13 der Datei user.h.

5.25.1.2 BUFFER_SIZE

```
#define BUFFER_SIZE 256
```

Definiert in Zeile 15 der Datei user.h.

5.25.1.3 BUFFER_SUCCESS

```
#define BUFFER_SUCCESS 1
```

Definiert in Zeile 14 der Datei user.h.

5.25.1.4 LED0

```
#define LED0 _LATB8
```

Definiert in Zeile 4 der Datei user.h.

5.25.1.5 LED1

```
#define LED1 _LATB9
```

Definiert in Zeile 5 der Datei user.h.

5.25.1.6 LED2

```
#define LED2 _LATB10
```

Definiert in Zeile 6 der Datei user.h.

5.25.1.7 LED3

```
#define LED3 _LATB11
```

Definiert in Zeile 7 der Datei user.h.

5.25.1.8 SENSOR_TIME

```
#define SENSOR_TIME 2
```

Definiert in Zeile 16 der Datei user.h.

5.25.1.9 T0

```
#define T0 !_RG12
```

Definiert in Zeile 8 der Datei user.h.

5.25.1.10 T1

```
#define T1 !_RG13
```

Definiert in Zeile 9 der Datei user.h.

5.25.1.11 T2

```
#define T2 !_RG14
```

Definiert in Zeile 10 der Datei user.h.

5.25.1.12 T3

```
#define T3 !_RG15
```

Definiert in Zeile 11 der Datei user.h.

5.25.2 Dokumentation der Funktionen

5.25.2.1 InitApp()

```
void InitApp (
             void  )
```

Definiert in Zeile 26 der Datei user.c.

Hier ist ein Graph der zeigt, wo diese Funktion aufgerufen wird:

5.26 user.h

gehe zur Dokumentation dieser Datei

```
00001
00002 /* User Level #define Macros                                              */
00003
00004 #define LED0 _LATB8
00005 #define LED1 _LATB9
00006 #define LED2 _LATB10
00007 #define LED3 _LATB11
00008 #define T0 !_RG12
00009 #define T1 !_RG13
00010 #define T2 !_RG14
00011 #define T3 !_RG15
00012
00013 #define BUFFER_FAIL     0
00014 #define BUFFER_SUCCESS  1
00015 #define BUFFER_SIZE 256
00016 #define SENSOR_TIME 2
00017 /* TODO Application specific user parameters used in user.c may go here */
00018
00019
00020 /* User Function Prototypes                                               */
00021
00023 /* TODO User level functions prototypes (i.e. InitApp) go here */
00024
00025 void InitApp(void);         /* I/O and Peripheral Initialization */
00026
00027
```

5.27 Code Style.markdown-Dateireferenz

5.28 Dokumentation.markdown-Dateireferenz

Index