

Embedded 2 Dokumentation

Erzeugt von Doxygen 1.9.3

1 Programmtdokumentation	1
2 Code-Style	3
2.0.1 Einrückung, Klammern und Formatierung	3
2.0.2 Kommentare	3
2.0.3 Bezeichner / Namen von Variablen, Funktionen & Konstanten	4
2.0.4 Bibliotheken	4
2.0.5 Doxygen	4
3 Aufgabe I2C Lichtsensor	5
4 Erweiterung I2C Lichtsensor Callback	7
5 Aufgabe LCD Display GPIO	9
6 Aufgabe LCD Display PMP	11
7 Datenstruktur-Verzeichnis	13
7.1 Datenstrukturen	13
8 Datei-Verzeichnis	15
8.1 Auflistung der Dateien	15
9 Datenstruktur-Dokumentation	17
9.1 Buffer Strukturreferenz	17
9.1.1 Ausführliche Beschreibung	17
9.1.2 Dokumentation der Felder	17
9.1.2.1 data	18
9.1.2.2 read	18
9.1.2.3 write	18
9.2 Buffer_I2C_FSM Strukturreferenz	18
9.2.1 Ausführliche Beschreibung	19
9.2.2 Dokumentation der Felder	19
9.2.2.1 data	19
9.2.2.2 read	20
9.2.2.3 write	20
9.3 I2C_struct Strukturreferenz	20
9.3.1 Ausführliche Beschreibung	21
9.3.2 Dokumentation der Felder	21
9.3.2.1 address	21
9.3.2.2 callback	21
9.3.2.3 ID	21
9.3.2.4 num_read	21
9.3.2.5 num_write	22
9.3.2.6 readbuf	22

9.3.2.7 status	22
9.3.2.8 writebuf	22
10 Datei-Dokumentation	23
10.1 configuration_bits.c-Dateireferenz	23
10.2 configuration_bits.c	23
10.3 I2C.c-Dateireferenz	24
10.3.1 Dokumentation der Funktionen	26
10.3.1.1 dolI2C()	26
10.3.1.2 exchangel2C()	26
10.3.1.3 FSM_Adresse_Read()	27
10.3.1.4 FSM_Adresse_Write()	28
10.3.1.5 FSM_Idle()	29
10.3.1.6 FSM_RECV_EN()	30
10.3.1.7 FSM_Repeated_Start()	31
10.3.1.8 FSM_Start()	32
10.3.1.9 FSM_Stop()	33
10.3.1.10 get_I2C_struct_FIFO()	33
10.3.1.11 I2C_LightSens_Callback()	34
10.3.1.12 I2C_TempSens_Callback()	35
10.3.1.13 initI2C()	36
10.3.1.14 put_I2C_struct_FIFO()	37
10.3.2 Variablen-Dokumentation	38
10.3.2.1 FIFO_I2C	38
10.3.2.2 I2C_test_struct	38
10.4 I2C.c	39
10.5 I2C.h-Dateireferenz	43
10.5.1 Makro-Dokumentation	45
10.5.1.1 I2C_SCL	45
10.5.1.2 I2C_SCL_TRIS	45
10.5.1.3 I2C_SDA	46
10.5.1.4 I2C_SDA_TRIS	46
10.5.2 Dokumentation der benutzerdefinierten Typen	46
10.5.2.1 I2C_Callback_t	46
10.5.2.2 StateFunc	46
10.5.3 Dokumentation der Aufzählungstypen	46
10.5.3.1 i2c_status_t	46
10.5.4 Dokumentation der Funktionen	47
10.5.4.1 dolI2C()	47
10.5.4.2 exchangel2C()	47
10.5.4.3 FSM_Adresse_Read()	48
10.5.4.4 FSM_Adresse_Write()	49

10.5.4.5 FSM_Idle()	50
10.5.4.6 FSM_RECV_EN()	51
10.5.4.7 FSM_Repeated_Start()	52
10.5.4.8 FSM_Start()	53
10.5.4.9 FSM_Stop()	54
10.5.4.10 I2C_LightSens_Callback()	54
10.5.4.11 I2C_TempSens_Callback()	55
10.5.4.12 initI2C()	56
10.5.5 Variablen-Dokumentation	57
10.5.5.1 FIFO_I2C	57
10.5.5.2 I2C_test_struct	57
10.5.5.3 latest_temperatur	57
10.5.5.4 read_data_buffer_light	57
10.5.5.5 read_data_buffer_temp	58
10.5.5.6 status_licht	58
10.5.5.7 status_temperatur	58
10.5.5.8 write_data_buffer_light	58
10.5.5.9 write_data_buffer_temp	58
10.6 I2C.h	59
10.7 interrupts.c-Dateireferenz	60
10.7.1 Dokumentation der Funktionen	60
10.7.1.1 _T1Interrupt()	60
10.8 interrupts.c	61
10.9 lcd_gpio.c-Dateireferenz	62
10.9.1 Dokumentation der Funktionen	62
10.9.1.1 lcd_clear()	62
10.9.1.2 lcd_get_status()	63
10.9.1.3 lcd_init()	64
10.9.1.4 lcd_set_pos()	64
10.9.1.5 lcd_write_data()	65
10.9.1.6 waitForBusyLCD()	66
10.9.1.7 writeStrLCD()	66
10.10 lcd_gpio.c	67
10.11 lcd_gpio.h-Dateireferenz	70
10.11.1 Makro-Dokumentation	71
10.11.1.1 __delay_cycles	71
10.11.1.2 CURSOR_OR_DISPLAY	71
10.11.1.3 LCD_CMD_INIT	72
10.11.1.4 LCD_DATA	72
10.11.1.5 LCD_DISPLAY_CLEAR	72
10.11.1.6 LCD_DISPLAY_HOME	72
10.11.1.7 LCD_DISPLAY_OFF	72

10.11.1.8 LCD_DISPLAY_ON	72
10.11.1.9 LCD_ENABLE	73
10.11.1.10 LCD_ENTRY_MODE	73
10.11.1.11 LCD_FUNCTION_SET	73
10.11.1.12 LCD_R_W	73
10.11.1.13 LCD_RS	73
10.11.1.14 READ_BUSY_FLAG	73
10.11.2 Dokumentation der Funktionen	74
10.11.2.1 lcd_clear()	74
10.11.2.2 lcd_get_status()	74
10.11.2.3 lcd_init()	74
10.11.2.4 lcd_set_pos()	75
10.11.2.5 lcd_write_data()	75
10.11.2.6 waitForBusyLCD()	75
10.11.2.7 writeStrLCD()	75
10.12 lcd_gpio.h	76
10.13 main.c-Dateireferenz	77
10.13.1 Makro-Dokumentation	78
10.13.1.1 HEARTBEAT_MS	78
10.13.1.2 MAIN	78
10.13.2 Dokumentation der Funktionen	78
10.13.2.1 display_temp_load()	78
10.13.2.2 display_UART_RX()	79
10.13.2.3 get_Light()	79
10.13.2.4 get_Temperatur()	79
10.13.2.5 main()	80
10.13.2.6 measureProcesstime()	80
10.13.2.7 print_sensor_values()	80
10.13.2.8 send_lcd_status()	81
10.13.3 Variablen-Dokumentation	81
10.13.3.1 latest_cpu_load	81
10.13.3.2 latest_temperatur	81
10.13.3.3 read_data_buffer_light	81
10.13.3.4 read_data_buffer_temp	82
10.13.3.5 received_UART	82
10.13.3.6 status_licht	82
10.13.3.7 status_temperatur	82
10.13.3.8 UART_RX_count	82
10.13.3.9 write_data_buffer_light	82
10.13.3.10 write_data_buffer_temp	83
10.14 main.c	83
10.15 main_less.c-Dateireferenz	86

10.15.1 Makro-Dokumentation	87
10.15.1.1 BAUDRATE	88
10.15.1.2 BRGVAL	88
10.15.1.3 BUFFER_FAIL	88
10.15.1.4 BUFFER_SIZE	88
10.15.1.5 BUFFER_SUCCESS	88
10.15.1.6 HEARTBEAT_MS	88
10.15.1.7 I2C_SCL	89
10.15.1.8 I2C_SCL_TRIS	89
10.15.1.9 I2C_SDA	89
10.15.1.10 I2C_SDA_TRIS	89
10.15.2 Dokumentation der benutzerdefinierten Typen	89
10.15.2.1 StateFunc	89
10.15.3 Dokumentation der Funktionen	89
10.15.3.1 _T1Interrupt()	90
10.15.3.2 _U1TXInterrupt()	90
10.15.3.3 delay_ms()	90
10.15.3.4 FSM2_ACK_Receive()	91
10.15.3.5 FSM2_Adresse()	92
10.15.3.6 FSM2_Data_Receive()	92
10.15.3.7 FSM2_Idle()	93
10.15.3.8 FSM2_Start()	94
10.15.3.9 FSM2_Stop()	94
10.15.3.10 getcFIFO_TX()	95
10.15.3.11 init_ms_t4()	95
10.15.3.12 init_timer1()	95
10.15.3.13 initI2C()	96
10.15.3.14 initUART()	96
10.15.3.15 main()	97
10.15.3.16 putcFIFO_TX()	98
10.15.3.17 putcUART()	98
10.15.3.18 putsUART()	98
10.15.3.19 Temp_FSM2()	99
10.15.4 Variablen-Dokumentation	99
10.15.4.1 data	99
10.15.4.2 DELAY_ANPASSUNG	100
10.15.4.3 FIFO	100
10.16 main_less.c	100
10.17 PMP.c-Dateireferenz	105
10.17.1 Dokumentation der Funktionen	106
10.17.1.1 initPMP()	106
10.17.1.2 lcd_clear()	106

10.17.1.3 lcd_get_status()	107
10.17.1.4 lcd_set_pos()	107
10.17.1.5 lcd_write_data()	108
10.17.1.6 waitForBusyLCD()	109
10.17.1.7 writeStrLCD()	109
10.18 PMP.c	110
10.19 PMP.h-Dateireferenz	112
10.19.1 Makro-Dokumentation	114
10.19.1.1 __delay_cycles	114
10.19.1.2 CURSOR_OR_DISPLAY	114
10.19.1.3 LCD_CMD_INIT	114
10.19.1.4 LCD_DATA	115
10.19.1.5 LCD_DISPLAY_CLEAR	115
10.19.1.6 LCD_DISPLAY_HOME	115
10.19.1.7 LCD_DISPLAY_OFF	115
10.19.1.8 LCD_DISPLAY_ON	115
10.19.1.9 LCD_ENABLE	115
10.19.1.10 LCD_ENTRY_MODE	116
10.19.1.11 LCD_FUNCTION_SET	116
10.19.1.12 LCD_R_W	116
10.19.1.13 LCD_RS	116
10.19.1.14 READ_BUSY_FLAG	116
10.19.2 Dokumentation der Funktionen	116
10.19.2.1 initPMP()	117
10.19.2.2 lcd_clear()	117
10.19.2.3 lcd_get_status()	118
10.19.2.4 lcd_set_pos()	119
10.19.2.5 lcd_write_data()	119
10.19.2.6 waitForBusyLCD()	120
10.19.2.7 writeStrLCD()	121
10.20 PMP.h	121
10.21 system.c-Dateireferenz	122
10.21.1 Makro-Dokumentation	123
10.21.1.1 CYCLES_PER_MIKROSECONDS	123
10.21.1.2 CYCLES_PER_MILLISECONDS	123
10.21.2 Dokumentation der Funktionen	123
10.21.2.1 ConfigureOscillator()	124
10.21.2.2 delay_ms()	124
10.21.2.3 delay_us()	124
10.21.2.4 init_ms_t4()	125
10.21.2.5 init_t2_t3()	125
10.21.2.6 init_timer1()	125

10.21.3 Variablen-Dokumentation	126
10.21.3.1 DELAY_ANPASSUNG	126
10.22 system.c	126
10.23 system.h-Dateireferenz	128
10.23.1 Makro-Dokumentation	129
10.23.1.1 FCY	129
10.23.1.2 SYS_FREQ	129
10.23.2 Dokumentation der Funktionen	129
10.23.2.1 ConfigureOscillator()	129
10.23.2.2 delay_ms()	129
10.23.2.3 init_ms_t4()	130
10.23.2.4 init_t2_t3()	130
10.23.2.5 init_timer1()	131
10.23.3 Variablen-Dokumentation	131
10.23.3.1 DELAY_ANPASSUNG	131
10.24 system.h	132
10.25 traps.c-Dateireferenz	132
10.25.1 Dokumentation der Funktionen	133
10.25.1.1 _AddressError()	133
10.25.1.2 _DefaultInterrupt()	133
10.25.1.3 _MathError()	133
10.25.1.4 _OscillatorFail()	133
10.25.1.5 _StackError()	134
10.26 traps.c	134
10.27 UART.c-Dateireferenz	136
10.27.1 Dokumentation der Funktionen	137
10.27.1.1 _U1RXInterrupt()	137
10.27.1.2 _U1TXInterrupt()	137
10.27.1.3 getcFIFO_RX()	137
10.27.1.4 getcFIFO_TX()	138
10.27.1.5 initUART()	138
10.27.1.6 putcFIFO_RX()	138
10.27.1.7 putcFIFO_TX()	139
10.27.1.8 putcUART()	139
10.27.1.9 putsUART()	140
10.27.2 Variablen-Dokumentation	140
10.27.2.1 FIFO	140
10.27.2.2 FIFO_RX	140
10.28 UART.c	140
10.29 UART.h-Dateireferenz	143
10.29.1 Makro-Dokumentation	144
10.29.1.1 BAUDRATE	144

10.29.1.2 BRGVAL	144
10.29.2 Dokumentation der Funktionen	144
10.29.2.1 getcFIFO_RX()	145
10.29.2.2 getcFIFO_TX()	145
10.29.2.3 initUART()	145
10.29.2.4 putcFIFO_RX()	146
10.29.2.5 putcFIFO_TX()	146
10.29.2.6 putsUART()	147
10.29.3 Variablen-Dokumentation	147
10.29.3.1 FIFO	147
10.29.3.2 FIFO_RX	147
10.29.3.3 received_UART	148
10.29.3.4 UART_RX_count	148
10.30 UART.h	148
10.31 user.c-Dateireferenz	149
10.31.1 Dokumentation der Funktionen	149
10.31.1.1 InitApp()	149
10.31.1.2 setLED()	150
10.32 user.c	150
10.33 user.h-Dateireferenz	150
10.33.1 Makro-Dokumentation	151
10.33.1.1 BUFFER_FAIL	152
10.33.1.2 BUFFER_SIZE	152
10.33.1.3 BUFFER_SUCCESS	152
10.33.1.4 LED0	152
10.33.1.5 LED1	152
10.33.1.6 LED2	152
10.33.1.7 LED3	153
10.33.1.8 SENSOR_TIME	153
10.33.1.9 T0	153
10.33.1.10 T1	153
10.33.1.11 T2	153
10.33.1.12 T3	153
10.33.2 Dokumentation der Funktionen	154
10.33.2.1 InitApp()	154
10.34 user.h	154
10.35 Code Style.markdown-Dateireferenz	154
10.36 Dokumentation.markdown-Dateireferenz	154
Index	155

Kapitel 1

Programmdokumentation

Die Dokumentation erfolgt getrennt nach den jeweiligen Aufgaben

Kapitel 2

Code-Style

Hier wird der aktuelle Coding Style hinterlegt.

2.0.1 Einrückung, Klammern und Formatierung

Der Code sollte möglichst einfach lesbar sein und gut strukturiert sein. Die geschwungenen Klammern von Blöcken wie z.B. If-Blöcken stehen immer in einer neuen, alleinstehenden Zeile. Blöcke werden eingerückt.

```
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.

2.0.2 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 über mehrere Zeilen erstreckt, wird dieser mit /* */ begrenzt. Andernfalls reichen die zwei doppelten Slashes //. 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*/
```

2.0.3 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 können auch Abkürzungen enthalten. Variablen- und Funktionen-Bezeichner werden durch Unterstriche getrennt und sind klein geschrieben. (snake_case) Konstanten hingegen werden groß geschrieben.

```
//Konstanten
int DELAY_MS=4;

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

2.0.4 Bibliotheken

Bibliotheken werden generell immer am Anfang der Datei eingebunden.

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

2.0.5 Doxygen

Einleitung mit /** Nur die nötigsten Tags verwenden.

```
/**  
 * @brief Verzögerung (ms)  
 * Verzögerungsfunktion, blockierend  
 * @param milliseconds Anzahl der zu verzögerten Millisekunden  
 * @return ...  
 */
```

Kapitel 3

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 grundlegende 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 `exchangel2C()` getriggert, welche als 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

Die grundlegende Funktion ist wie folgt:

Im Interrupt des Timer1 `_T1Interrupt()` wird zyklisch eine Anfrage mit der Funktion `exchangel2C()` getätig. In der Superloop befindet sich die Funktion `doI2C()`, welche die FSM enthält. Außerdem befindet sich dort die Funktion `print_sensor_values()`, welche die Sensor-Wert nach erfolgreicher Abarbeitung der Anfrage ausgibt.

Die Funktionen in der I2C funktionieren wie folgt:

In der Funktion `put_I2C_struct_FIFO()` wird eine I2C-Anfrage im I2C-FIFO abgelegt. Die Funktion `get_I2C_struct_FIFO()` entnimmt die I2C-Anfrage aus dem I2C-FIFO. Mit der `initI2C()` wird die I2C-Kommunikation initialisiert.

Anschließend kommen die Funktionen der FSM:

Zu Beginn die Funktion `FSM_Idle()`. Diese kopiert die Anfrage aus dem FIFO und leitet die Start-Sequenz ein. Wenn es eine Anfrage gibt, werden die Startbedingungen weitergeleitet. Mit der Funktion `FSM_Start()` wird das Trancieve-Register mit der Adresse beschrieben. Entweder wird geschrieben oder gelesen. Wenn geschrieben wird, wird die Funktion `FSM_Adresse_Write()` aufgerufen und die zu übertragenden Daten werden in das Tranceive-Register geschrieben. Wenn es nichts mehr zu senden gibt, dann kommt es zum Restart und die Funktion `FSM_Repeated_Start()` wird aufgerufen. Hier wird wieder das Tranceive-Register mit der Adresse beschrieben. Wenn der Restart erfolgreich war, wird geschaut ob noch gelesen werden kann. Hierbei wird die Funktion `FSM_Adresse_Read()` aufgerufen. Hier wird das Lesen der Daten des Slaves initiiert. Gibt es noch etwas zum Lesen wird die Funktion `FSM_RECV_EN()` aufgerufen. Diese liest das Receive Register aus und bestätigt dies mit einem ACK oder NACK. Treten Fehlerfälle beim Lesen oder Schreiben auf, wird eine Stopp-Bedingung eingeleitet. Dies hat zur Folge das die Funktion `FSM_Stop()` aufgerufen wird. Diese verursacht die Rückkehr in den Idle-State.

Der Status der Anfragen wird innerhalb der FSM gesetzt. Falls der Status auf Finished gesetzt wird, gibt `print_sensor_values()` die gemessenen Werte aus.

Kapitel 4

Erweiterung I2C Lichtsensor Callback

In dieser Aufgabe wurde die I2C-Funktionalität um die Möglichkeit erweitert, eine Callback Funktion zu definieren. Diese Funktion wird aufgerufen, sobald die FSM die Übertragung stoppt. Das kann sowohl im Normal-Fall als auch im Fehlerfall erfolgen. Die zwei Callback-Funktionen `I2C_TempSens_Callback()` und `I2C_LightSens_Callback()` geben dann jeweils die Temperatur oder den Lichtwert aus, falls der Status auf Finished gesetzt wurde. Die Funktionen müssen dabei vom Typ `I2C_Callback_t` sein, welcher die Weitergabe der benötigten Zegier und Daten erlaubt.

Kapitel 5

Aufgabe LCD Display GPIO

In dieser Aufgabe sollte das LCD Display in Betrieb genommen werden. Es wurde dafür die Bibliothek `lcd_gpio.h` geschrieben. Der Zugriff erfolgt dabei noch über die GPIOs. Um das LCD anzusteuern muss dieses vorher einmalig initialisiert werden. Dies geschieht in der Funktion `lcd_init()`.

Ist das LCD initialisiert, können über die Funktion `lcd_write_data()` Daten über den Bus versendet werden. Mit der Funktion `lcd_get_status()` wird das Busy-Flag und der Adress-Zähler abgefragt. Die Funktion `waitForBusyLCD()` liest das Busy-Flag und blockiert solange dies gesetzt ist. In den anderen Funktionen, welche einen Schreib-Zugriff implementieren wird immer zuerst die Funktion `waitForBusyLCD()` ausgeführt um zu gewährleisten, dass die zu sendenden Daten auch vom LCD interpretiert werden können.

Es wurde eine Funktion `writeStrLCD()` geschrieben, welche es erlaubt komplette Strings auf dem LCD anzuzeigen.

Weiter Funktionen sind `lcd_clear()`, welche das LCD zurücksetzt und den Inhalt löscht sowie `lcd_set_pos()`, welche den Cursor auf die gewünschte Position bewegt.

Um die geforderten Timings einzuhalten wurden zwei unterschiedliche Delay-Mechanismen verwendet: `__delay_cycles()` verzögert um die angegebenen Zyklen. In unserem Fall beträgt FCY 50MHZ also ist ein Taktzyklus 20ns lang. `__delay_us()` verzögert um die angegebenen Mikrosekunden.

In der SuperLoop wird die Funktion `display_temp_load()` jede ms aufgerufen und gibt in einem Takt von 3s abwechselnd die aktuelle Temperatur und Aulastung in der ersten Zeile des LCDs aus. Des Weiteren wird alle 3s die per UART empfangenen Daten in der zweiten Zeile des LCDs ausgegeben.

Hier zu sehen ist das Timing der LCD Signale. Als erstes wird hier die Funktion `lcd_set_pos()` ausgeführt. In dieser wird das Display auf die Home position gesetzt. Danach wird zyklisch das Busy-Flag abgefragt. Es dauert ca 0.4ms bis es nicht mehr gesetzt ist. Danach erfolgen Schreib-Zugriffe, welche jeweils ca. 30us andauern.

Kapitel 6

Aufgabe LCD Display PMP

In dieser Aufgabe sollte das LCD mit Hilfe des PMP-Moduls angesteuert werden. Es wurde hierfür die Bibliothek [PMP.h](#) geschrieben. Der Parallel Master Port (PMP) ist ein paralleles Kommunikationsmodul, das speziell für die Kommunikation mit einer Vielzahl von parallelen Geräten, wie zum Beispiel in unserem Fall ein LCD, verwendbar ist. Um die Ansteuerung des LCDs zu ermöglichen, wird vorab eine Initialisierung vorgenommen. Hierfür wurde die Funktion [initPMP\(\)](#) implementiert. Anschließend wurde darauf geachtet, dass dieselben Funktionen verwendet wurden wie schon bei der Ansteuerung mittels GPIO. Diese wurden größtenteils angepasst, damit die Ansteuerung über PMP funktioniert. Dadurch das wir dieselben Funktionen verwendet haben, kann man jederzeit problemlos in der Hauptschleife zwischen der Ansteuerung über PMP oder GPIO wechseln.

Kapitel 7

Datenstruktur-Verzeichnis

7.1 Datenstrukturen

Hier folgt die Aufzählung aller Datenstrukturen mit einer Kurzbeschreibung:

Buffer	17
Buffer_I2C_FSM	18
I2C_struct	
Datenstruktur für die Kapselung aller benötigten Variablen, welche für ein character basiertes FIFO benötigt werden	20

Kapitel 8

Datei-Verzeichnis

8.1 Auflistung der Dateien

Hier folgt die Aufzählung aller Dateien mit einer Kurzbeschreibung:

configuration_bits.c	23
I2C.c	24
I2C.h	43
interrupts.c	60
lcd_gpio.c	62
lcd_gpio.h	70
main.c	77
main_less.c	86
PMP.c	105
PMP.h	112
system.c	122
system.h	128
traps.c	132
UART.c	136
UART.h	143
user.c	149
user.h	150

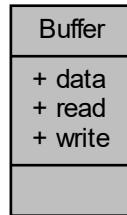
Kapitel 9

Datenstruktur-Dokumentation

9.1 Buffer Strukturreferenz

```
#include <UART.h>
```

Zusammengehörigkeiten von Buffer:



Datenfelder

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

9.1.1 Ausführliche Beschreibung

Definiert in Zeile 50 der Datei [main_less.c](#).

9.1.2 Dokumentation der Felder

9.1.2.1 data

```
uint8_t data
```

Definiert in Zeile 51 der Datei [main_less.c](#).

9.1.2.2 read

```
uint8_t read
```

Definiert in Zeile 52 der Datei [main_less.c](#).

9.1.2.3 write

```
uint8_t write
```

Definiert in Zeile 53 der Datei [main_less.c](#).

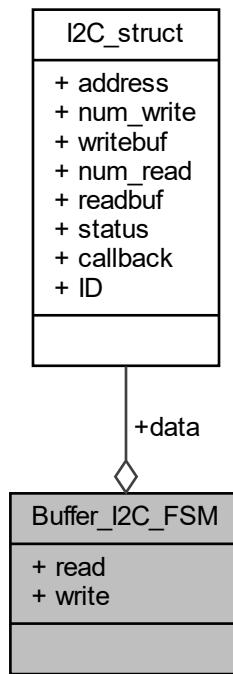
Die Dokumentation für diese Struktur wurde erzeugt aufgrund der Dateien:

- [main_less.c](#)
- [UART.h](#)

9.2 Buffer_I2C_FSM Strukturreferenz

```
#include <I2C.h>
```

Zusammengehörigkeiten von Buffer_I2C_FSM:



Datenfelder

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

9.2.1 Ausführliche Beschreibung

Definiert in Zeile 56 der Datei [I2C.h](#).

9.2.2 Dokumentation der Felder

9.2.2.1 data

`I2C_struct data [BUFFER_SIZE]`

Definiert in Zeile 58 der Datei [I2C.h](#).

9.2.2.2 `read`

```
uint8_t read
```

Definiert in Zeile 59 der Datei [I2C.h](#).

9.2.2.3 `write`

```
uint8_t write
```

Definiert in Zeile 60 der Datei [I2C.h](#).

Die Dokumentation für diese Struktur wurde erzeugt aufgrund der Datei:

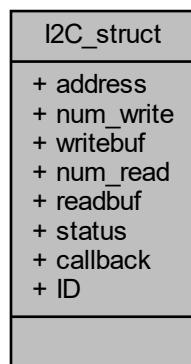
- [I2C.h](#)

9.3 I2C_struct Strukturreferenz

Datenstruktur für die Kapselung aller benötigten Variablen, welche für ein character basiertes FIFO benötigt werden.

```
#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`
- `I2C_Callback_t callback`
- `int16_t ID`

9.3.1 Ausführliche Beschreibung

Datenstruktur für die Kapselung aller benötigten Variablen, welche für ein character basiertes FIFO benötigt werden.

Diese Datenstruktur muss als globale Variable (z.B. mit Zusatz static) angelegt werden

Definiert in Zeile 43 der Datei [I2C.h](#).

9.3.2 Dokumentation der Felder

9.3.2.1 address

```
uint8_t address
```

Definiert in Zeile 45 der Datei [I2C.h](#).

9.3.2.2 callback

```
I2C_Callback_t callback
```

Definiert in Zeile 51 der Datei [I2C.h](#).

9.3.2.3 ID

```
int16_t ID
```

Definiert in Zeile 52 der Datei [I2C.h](#).

9.3.2.4 num_read

```
uint16_t num_read
```

Definiert in Zeile 48 der Datei [I2C.h](#).

9.3.2.5 num_write

```
uint16_t num_write
```

Definiert in Zeile 46 der Datei [I2C.h](#).

9.3.2.6 readbuf

```
uint8_t* readbuf
```

Definiert in Zeile 49 der Datei [I2C.h](#).

9.3.2.7 status

```
i2c_status_t* status
```

Definiert in Zeile 50 der Datei [I2C.h](#).

9.3.2.8 writebuf

```
uint8_t* writebuf
```

Definiert in Zeile 47 der Datei [I2C.h](#).

Die Dokumentation für diese Struktur wurde erzeugt aufgrund der Datei:

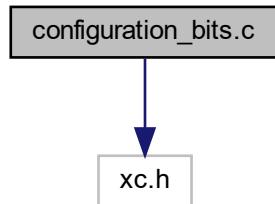
- [I2C.h](#)

Kapitel 10

Datei-Dokumentation

10.1 configuration_bits.c-Dateireferenz

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



10.2 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 WDTPS = 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 = OFF          // Watchdog Timer Enable bit (Watchdog timer always enabled)
00027
00028 // FPOR
00029 #pragma config FWRT = 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 JTGEN = 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

```

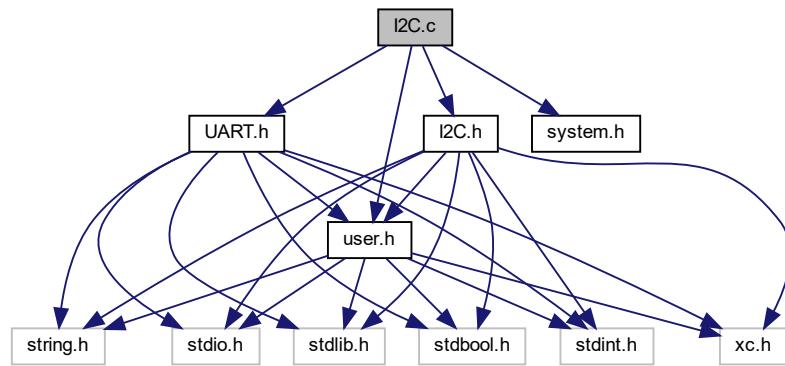
10.3 I2C.c-Dateireferenz

```

#include "I2C.h"
#include "UART.h"
#include "system.h"
#include "user.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, I2C_Callback_t callback, int16_t ID)`
Übergibt angeforderte I2C-Anfrage an das FIFO und liefert den aktuellen Status.
- `void doI2C ()`
Wird jede ms in der Superloop ausgeführt und beinhaltet 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 I2C_TempSens_Callback (uint8_t *readbuf, uint16_t num_read, i2c_status_t *status, int16_t ID)`
Callback-Funktion zum Augeben der Temperaturwerte.
- `void I2C_LightSens_Callback (uint8_t *readbuf, uint16_t num_read, i2c_status_t *status, int16_t ID)`
Callback-Funktion zum Augeben der Lichtwerte.

Variablen

- I2C_struct I2C_test_struct = {0,0,NULL,0,NULL,NULL}
- Buffer_I2C_FSM FIFO_I2C = {{},0,0}

10.3.1 Dokumentation der Funktionen

10.3.1.1 doI2C()

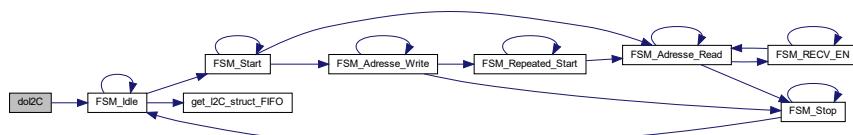
```
void doI2C (
    void )
```

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

Falls das FIFO neue Anfragen enthält wird die FSM getriggert.

Definiert in Zeile 117 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:



10.3.1.2 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,
    I2C_Callback_t callback,
    int16_t ID )
```

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

Parameter

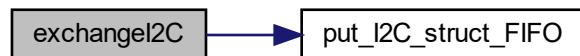
<i>address</i>	7 Bit Adresse des Slaves
<i>num_write</i>	Anzahl der zu sendenen Bytes, bei 0 keine Write Zugriff
<i>writebuf</i>	Zeiger auf zu schreibende Daten
<i>num_read</i>	Anzahl der zu lesenden Bytes, bei 0 keine Read Zugriff
<i>readbuf</i>	Zeiger auf Bereich, in welchem Daten abgespeichert werden sollen
<i>status</i>	Zeiger, um aktuellen Status zurückzugeben
<i>callback</i>	Zeiger auf eine Callback-Funktion, welche nach Abschluss der Anfrage aufgerufen werden soll
<i>ID</i>	ID zur Identifikation der Anfrage

Rückgabewerte

<i>1,Anforderung</i>	wurde angenommen, die FSM wird getriggert
<i>0,FSM</i>	ist beschäftigt, Anforderung kann nicht angeommen werden

Definiert in Zeile 96 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:



10.3.1.3 FSM_Adresse_Read()

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

Initiiert das Lesen der Daten des Slaves.

Parameter

<code>count</code>	Zaelervariable
--------------------	----------------

Rückgabe

`FSM_Stop`, sobald ein Fehler bei der Kommunikation auftritt, z.B kein ACK vom Slave. Es wird die Callback Funktion aufgerufen

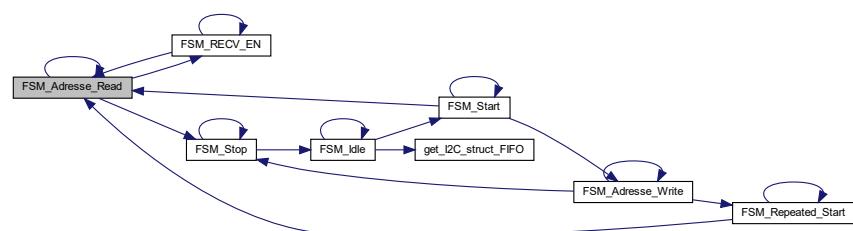
`FSM_RECV_EN`, sobald der Empfangsmodus für I2C aktiviert wurde

`FSM_Stop`, sobald die Stop-Bedigungen an die Pins SDAx und SCLx weitergeleitet wurden. Es wird die Callback Funktion aufgerufen

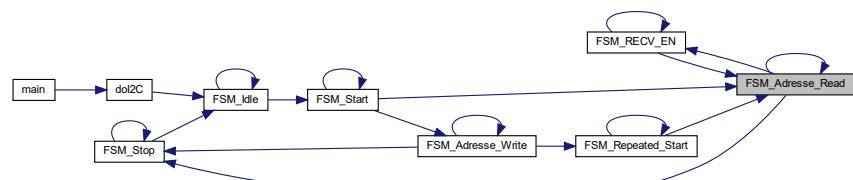
`FSM_Adresse_Read`, wenn kein ACK vom Slave erhalten oder das Bit der ACK-Sequenz nicht freigegeben ist

Definiert in Zeile 294 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:



10.3.1.4 `FSM_Adresse_Write()`

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

Schreibt die zu übertragende Daten in das Tranceive-Register.

Parameter

<i>count</i>	Zaelervariable
--------------	----------------

Rückgabe

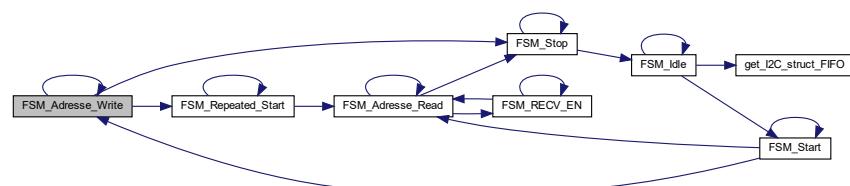
FSM_Stop, sobald ein Fehler bei der Kommunikation auftritt, z.B kein ACK vom Slave. Es wird die Callback Funktion aufgerufen

FSM_Adresse_Write, sobald keine Bytes mehr zu senden gibt

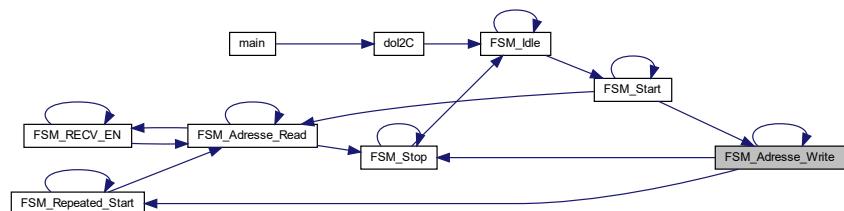
FSM_Repeated_Start, sobald die Bedingungen für den wiederholten Start an die Pins SDAx und SClx weitergeleitet wurde.

Definiert in Zeile 225 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:

**10.3.1.5 FSM_Idle()**

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

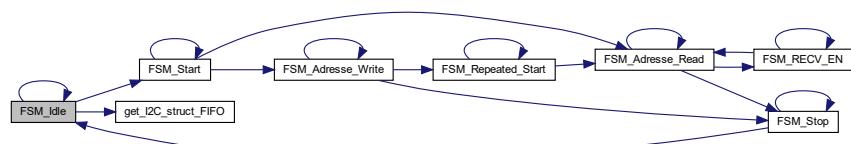
Kopiert die Anfrage aus dem FIFO und leitet Start-Sequenz ein.

Rückgabe

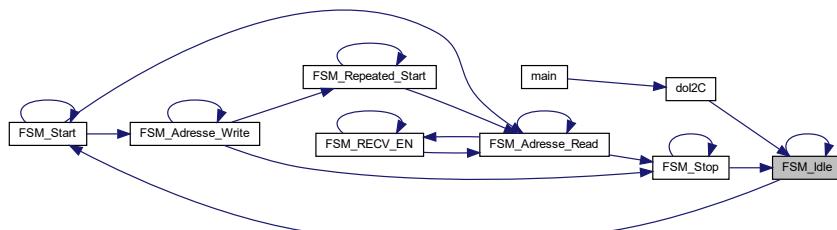
FSM_Start, sobald die Startbedingungen an die Pins SDAx und SCLx weitergeleitet worden sind

Definiert in Zeile 173 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:



10.3.1.6 FSM_RECV_EN()

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

Auslesen des Receive Registers und Bestätigung mit ACK bzw.

NACK

Parameter

count	Zaelervariable
-------	----------------

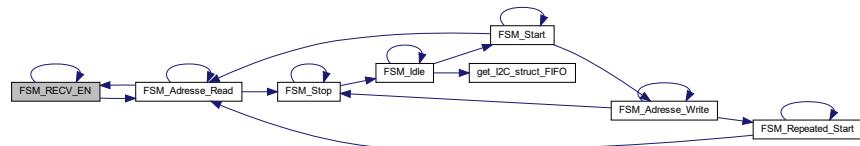
Rückgabe

FSM_Adresse_Read, sobald die Acknowledge-Sequenz an den Pins SDAx und SCLx initiiert wurde und das ACKDT Datenbit übertragen wurde

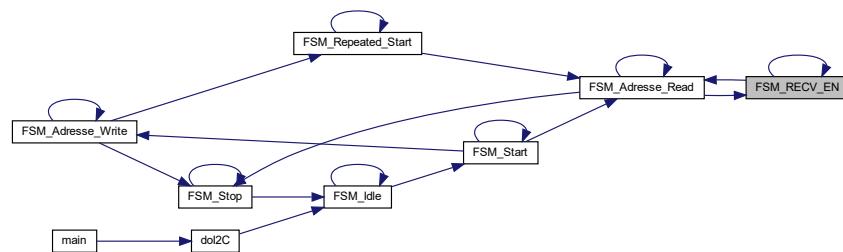
FSM_RECV_EN, Wenn die Empfangssequenz nicht ausgeführt wurde

Definiert in Zeile 356 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:



10.3.1.7 `FSM_Repeated_Start()`

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

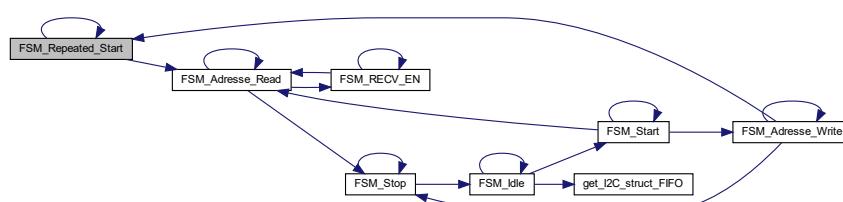
Leitet einen Repeated Start ein und beschreibt das Tranceive Register mit der Adresse.

Rückgabe

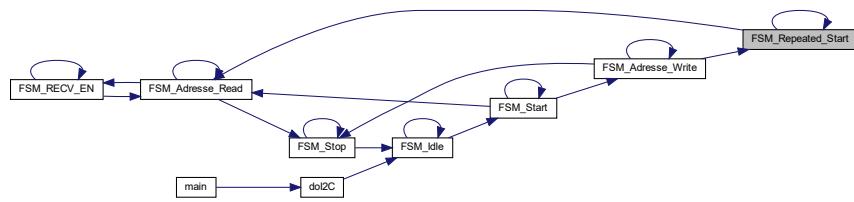
`FSM_Adresse_Read`, sobald es einen Restart gibt

Definiert in Zeile 271 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:



10.3.1.8 `FSM_Start()`

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

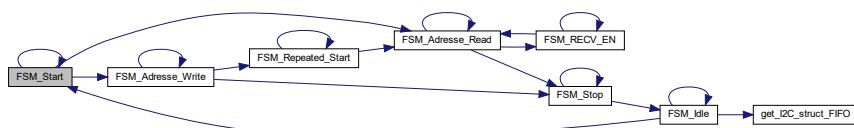
Beschreibt das Trancieve-Register mit der Adresse.

Rückgabe

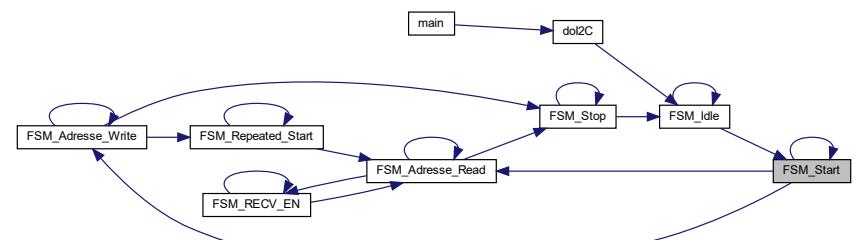
`FSM_Adresse_Write`, sobald geschrieben werden kann @retrun `FSM_Adresse_Read`, sobald gelesen werden kann

Definiert in Zeile 192 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:



10.3.1.9 FSM_Stop()

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

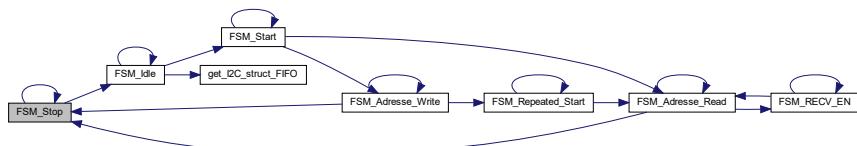
Überprüfung auf Abschluss der Stop-Sequenz und Rückkehr in den Idle-State.

Rückgabe

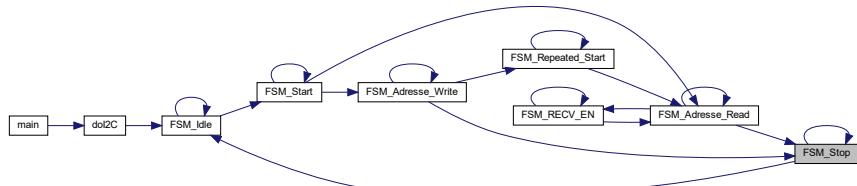
FSM_Idle, wenn die Stop-Bedingungen erfolgreich an den Pins SDAx und SCLx weitergeleitet wurden
 FSM_Stop, wenn keine Stop-Bedingungen weitergeleitet wurden

Definiert in Zeile 388 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:



10.3.1.10 get_I2C_struct_FIFO()

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

Entnimmt I2C-Anfrage aus dem I2C-FIFO.

Parameter

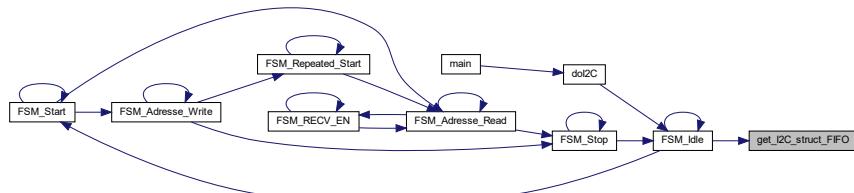
<code>*s</code>	Zeiger auf I2C-Anfrage in Form eines Structs des Typs I2C_struct
-----------------	--

Rückgabewerte

<i>BUFFER_FAIL</i>	im Fehlerfall
<i>ansonsten</i>	<i>BUFFER_SUCCESS</i> , wenn erfolgreich

Definiert in Zeile 59 der Datei [I2C.c](#).

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



10.3.1.11 I2C_LightSens_Callback()

```
void I2C_LightSens_Callback (
    uint8_t * readbuf,
    uint16_t num_read,
    i2c_status_t * status,
    int16_t ID )
```

Callback-Funktion zum Augeben der Lichtwerte.

Parameter

<i>readbuf</i>	Zeiger auf die ausgelesenen Daten
<i>num_read</i>	Anzahl der ausgelesenen Bytes
<i>status</i>	Status mit welchem die FSM die Callback-Funktion aufgerufen hat. Im Fall Error wird eine Fehlermeldung ausgegeben. Im Fall Finished werden die Daten interpretiert und ausgegeben.
<i>ID</i>	

Definiert in Zeile 450 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:



10.3.1.12 I2C_TempSens_Callback()

```
void I2C_TempSens_Callback (
    uint8_t * readbuf,
    uint16_t num_read,
    i2c_status_t * status,
    int16_t ID )
```

Callback-Funktion zum Augeben der Temperaturwerte.

Parameter

<i>readbuf</i>	Zeiger auf die ausgelesenen Daten
<i>num_read</i>	Anzahl der ausgelesenen Bytes
<i>status</i>	Status mit welchem die FSM die Callback-Funktion aufgerufen hat. Im Fall Error wird eine Fehlermeldung ausgegeben. Im Fall Finished werden die Daten interpretiert und ausgegeben.
<i>ID</i>	

Definiert in Zeile 408 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:



10.3.1.13 initI2C()

```
void initI2C (
    void )
```

Initialisiert die I2C-Kommunikation.

Einschalten I2C mit eigenem Workaround, I2C Peripheriemodul kann hier leider nicht verwendet werden. Pins wie einen Open-Kollektor-Treiber verwenden, d.h. 1 - hochohmig, 0 wird getrieben

Aktiviert das I2C Modul und konfiguriert die Pins SDAx und SCLx als serielle PORT-Pins.

Definiert in Zeile [126](#) 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:



10.3.1.14 put_I2C_struct_FIFO()

```
int16_t put_I2C_struct_FIFO (
```

`I2C_struct s)`

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

Parameter

<code>s</code>	I2C-Anfrage in Form eines Structs des Typs I2C_struct
----------------	---

Rückgabewerte

<code>BUFFER_FAIL</code>	im Fehlerfall
<code>ansonsten</code>	<code>BUFFER_SUCCESS</code> , wenn erfolgreich

Definiert in Zeile 30 der Datei [I2C.c](#).

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



10.3.2 Variablen-Dokumentation

10.3.2.1 FIFO_I2C

```
Buffer_I2C_FSM FIFO_I2C = {{}, 0, 0}
```

Definiert in Zeile 18 der Datei [I2C.c](#).

10.3.2.2 I2C_test_struct

```
I2C_struct I2C_test_struct = {0, 0, NULL, 0, NULL, NULL}
```

Definiert in Zeile 16 der Datei [I2C.c](#).

10.4 I2C.c

[gehe zur Dokumentation dieser Datei](#)

```

00001
00002 /* Files to Include
00003
00005 /* Beinhaltet Konstanten, Typdefs, globale Variablen und Prototypen für main */
00006 #include "I2C.h"
00007 #include "UART.h"
00008
00009 #include "system.h"      /* System - Funktion/Parameter
00010 #include "user.h"        /* Benutzer - Funktion/Parameter
00011
00012
00013 /* Global Variable Declaration
00014
00016 I2C_struct I2C_test_struct = {0,0,NULL,0,NULL,NULL};
00017
00018 Buffer_I2C_FSM FIFO_I2C = {{},0,0}; //FIFO für die I2C FSM
00019
00020
00021 /* Funktionen
00022
00030 int16_t put_I2C_struct_FIFO(I2C_struct s)
00031 {
00032     _GIE=0;
00033     if ( ( FIFO_I2C.write + 1 == FIFO_I2C.read ) ||
00034         ( FIFO_I2C.read == 0 && FIFO_I2C.write + 1 == BUFFER_SIZE ) )
00035     {
00036         _GIE=1;
00037         return BUFFER_FAIL; // voll
00038     }
00039
00040     FIFO_I2C.data[FIFO_I2C.write] = s;
00041
00042     FIFO_I2C.write++;
00043     if (FIFO_I2C.write >= BUFFER_SIZE)
00044     {
00045         FIFO_I2C.write = 0;
00046     }
00047     _GIE=1;
00048     return BUFFER_SUCCESS;
00049
00050 } /* put_I2C_struct_FIFO() */
00051
00052
00059 int16_t get_I2C_struct_FIFO(volatile I2C_struct *s)
00060 {
00061     _GIE=0;
00062     if (FIFO_I2C.read == FIFO_I2C.write)
00063     {
00064         _GIE=1;
00065         return BUFFER_FAIL;
00066     }
00067     *s = FIFO_I2C.data[FIFO_I2C.read];
00068
00069     FIFO_I2C.read++;
00070     if (FIFO_I2C.read >= BUFFER_SIZE)
00071     {
00072         FIFO_I2C.read = 0;
00073     }
00074     _GIE=1;
00075     return BUFFER_SUCCESS;
00076
00077 } /* get_I2C_struct_FIFO() */
00078
00079
00096 int16_t exchangeI2C(uint8_t address, uint16_t num_write, uint8_t *writebuf, uint16_t num_read, uint8_t_t
00097 *readbuf, i2c_status_t *status, I2C_Callback_t callback, int16_t ID)
00098 {
00099     I2C_struct temporary_struct = {address,num_write,writebuf,num_read,readbuf,status,callback,ID};
00100     if (put_I2C_struct_FIFO(temporary_struct)) //Erfolgreich in FIFO abgelegt?
00101     {
00102         return 1;
00103     }
00104     else
00105     {
00106         return 0;
00107     }
00108
00109 } /* exchangeI2C() */
00110
00111
00117 void doI2C()

```

```

00118 {
00119     static StateFunc statefunc = FSM_Idle;
00120     statefunc = (StateFunc) (*statefunc)();
00121 } /* doI2C() */
00122
00123 void initI2C()
00124 {
00125     I2C2CONbits.A10M = 0;
00126     I2C2BRG = 245; //100kHz
00127
00128     I2C_SDA_TRIS = 1;
00129     I2C_SCL_TRIS = 1;
00130
00131     I2C_SDA = 0;
00132     I2C_SCL = 0;
00133
00134     int j;
00135     for (j=0; j<=9; j++) // takten bis min 1 Byte
00136     {
00137         I2C_SCL_TRIS = 0; delay_ms(1); // 5 us wären ausreichend ...100 kBaud
00138         I2C_SCL_TRIS = 1; delay_ms(1);
00139     }
00140
00141     // Start Condition senden
00142     I2C_SCL_TRIS = 0; delay_ms(1);
00143     I2C_SDA_TRIS = 0; delay_ms(1);
00144
00145     // Stop Condition senden
00146     I2C_SCL_TRIS = 1; delay_ms(1);
00147     I2C_SDA_TRIS = 1; delay_ms(1);
00148
00149     // Nun I2C erst anschalten
00150     _MI2C2IF = 0; //Interrupt falls noetig
00151     _MI2C2IE = 0;
00152
00153     I2C2CONbits.I2CEN = 1;
00154
00155 } /* initI2C() */
00156
00157
00158
00159 void *FSM_Idle(void)
00160 {
00161     if (get_I2C_struct_FIFO(&I2C_test_struct)) //Anfrage aus FIFO holen möglich
00162     {
00163         I2C2CONbits.SEN=1; // Leite Start-Bedingungen weiter
00164         return FSM_Start;
00165     }
00166
00167
00168
00169     else
00170     {
00171         return FSM_Idle;
00172     }
00173 } /* *FSM_Idle() */
00174
00175
00176
00177
00178
00179
00180
00181
00182
00183
00184 } /* *FSM_Idle() */
00185
00186
00187
00188 void *FSM_Start(void)
00189 {
00190
00191     if (I2C2CONbits.SEN==0) // Wenn Startbedingungen erfüllt wurden
00192     {
00193         if (I2C_test_struct.num_write>0) //Schreiben
00194         {
00195             I2C2TRN=(I2C_test_struct.address<<1);
00196             return FSM_Adresse_Write;
00197         }
00198
00199         else if (I2C_test_struct.num_read>0) //Lesen
00200         {
00201             I2C2TRN=(I2C_test_struct.address<<1) | 0b1;
00202             return FSM_Adresse_Read;
00203         }
00204
00205
00206
00207
00208
00209
00210     }
00211     return FSM_Start;
00212
00213 } /* *FSM_Start() */
00214
00215
00216
00217 void *FSM_Adresse_Write(void)
00218 {
00219     if(I2C2STATbits.TRSTAT==0) //Wenn erfolgreich übertragen
00220     {
00221         if (I2C2STATbits.ACKSTAT==1) // Leitet Stop-Bedingungen weiter
00222         {
00223             I2C2CONbits.PEN=1; //Fehler bei Kommunikation
00224             *I2C_test_struct.status=Error;
00225             if (I2C_test_struct.callback != NULL)
00226             {
00227                 I2C_test_struct.callback(I2C_test_struct.readbuf, I2C_test_struct.num_read,
00228                 I2C_test_struct.status, I2C_test_struct.ID);
00229             }
00230         }
00231     }
00232
00233
00234
00235
00236
00237
00238
00239
00240
00241
00242
00243
00244
00245
00246
00247
00248
00249
00250
00251
00252
00253
00254
00255
00256
00257
00258
00259
00260
00261
00262
00263
00264
00265
00266
00267
00268
00269
00270
00271
00272
00273
00274
00275
00276
00277
00278
00279
00280
00281
00282
00283
00284
00285
00286
00287
00288
00289
00290
00291
00292
00293
00294
00295
00296
00297
00298
00299
00300
00301
00302
00303
00304
00305
00306
00307
00308
00309
00310
00311
00312
00313
00314
00315
00316
00317
00318
00319
00320
00321
00322
00323
00324
00325
00326
00327
00328
00329
00330
00331
00332
00333
00334
00335
00336
00337
00338
00339
00340
00341
00342
00343
00344
00345
00346
00347
00348
00349
00350
00351
00352
00353
00354
00355
00356
00357
00358
00359
00360
00361
00362
00363
00364
00365
00366
00367
00368
00369
00370
00371
00372
00373
00374
00375
00376
00377
00378
00379
00380
00381
00382
00383
00384
00385
00386
00387
00388
00389
00390
00391
00392
00393
00394
00395
00396
00397
00398
00399
00400
00401
00402
00403
00404
00405
00406
00407
00408
00409
00410
00411
00412
00413
00414
00415
00416
00417
00418
00419
00420
00421
00422
00423
00424
00425
00426
00427
00428
00429
00430
00431
00432
00433
00434
00435
00436
00437
00438
00439
00440
00441
00442
00443
00444
00445
00446
00447
00448
00449
00450
00451
00452
00453
00454
00455
00456
00457
00458
00459
00460
00461
00462
00463
00464
00465
00466
00467
00468
00469
00470
00471
00472
00473
00474
00475
00476
00477
00478
00479
00480
00481
00482
00483
00484
00485
00486
00487
00488
00489
00490
00491
00492
00493
00494
00495
00496
00497
00498
00499
00500
00501
00502
00503
00504
00505
00506
00507
00508
00509
00510
00511
00512
00513
00514
00515
00516
00517
00518
00519
00520
00521
00522
00523
00524
00525
00526
00527
00528
00529
00530
00531
00532
00533
00534
00535
00536
00537
00538
00539
00540
00541
00542
00543
00544
00545
00546
00547
00548
00549
00550
00551
00552
00553
00554
00555
00556
00557
00558
00559
00560
00561
00562
00563
00564
00565
00566
00567
00568
00569
00570
00571
00572
00573
00574
00575
00576
00577
00578
00579
00580
00581
00582
00583
00584
00585
00586
00587
00588
00589
00590
00591
00592
00593
00594
00595
00596
00597
00598
00599
00600
00601
00602
00603
00604
00605
00606
00607
00608
00609
00610
00611
00612
00613
00614
00615
00616
00617
00618
00619
00620
00621
00622
00623
00624
00625
00626
00627
00628
00629
00630
00631
00632
00633
00634
00635
00636
00637
00638
00639
00640
00641
00642
00643
00644
00645
00646
00647
00648
00649
00650
00651
00652
00653
00654
00655
00656
00657
00658
00659
00660
00661
00662
00663
00664
00665
00666
00667
00668
00669
00670
00671
00672
00673
00674
00675
00676
00677
00678
00679
00680
00681
00682
00683
00684
00685
00686
00687
00688
00689
00690
00691
00692
00693
00694
00695
00696
00697
00698
00699
00700
00701
00702
00703
00704
00705
00706
00707
00708
00709
00710
00711
00712
00713
00714
00715
00716
00717
00718
00719
00720
00721
00722
00723
00724
00725
00726
00727
00728
00729
00730
00731
00732
00733
00734
00735
00736
00737
00738
00739
00740
00741
00742
00743
00744
00745
00746
00747
00748
00749
00750
00751
00752
00753
00754
00755
00756
00757
00758
00759
00760
00761
00762
00763
00764
00765
00766
00767
00768
00769
00770
00771
00772
00773
00774
00775
00776
00777
00778
00779
00780
00781
00782
00783
00784
00785
00786
00787
00788
00789
00790
00791
00792
00793
00794
00795
00796
00797
00798
00799
00800
00801
00802
00803
00804
00805
00806
00807
00808
00809
00810
00811
00812
00813
00814
00815
00816
00817
00818
00819
00820
00821
00822
00823
00824
00825
00826
00827
00828
00829
00830
00831
00832
00833
00834
00835
00836
00837
00838
00839
00840
00841
00842
00843
00844
00845
00846
00847
00848
00849
00850
00851
00852
00853
00854
00855
00856
00857
00858
00859
00860
00861
00862
00863
00864
00865
00866
00867
00868
00869
00870
00871
00872
00873
00874
00875
00876
00877
00878
00879
00880
00881
00882
00883
00884
00885
00886
00887
00888
00889
00890
00891
00892
00893
00894
00895
00896
00897
00898
00899
00900
00901
00902
00903
00904
00905
00906
00907
00908
00909
00910
00911
00912
00913
00914
00915
00916
00917
00918
00919
00920
00921
00922
00923
00924
00925
00926
00927
00928
00929
00930
00931
00932
00933
00934
00935
00936
00937
00938
00939
00940
00941
00942
00943
00944
00945
00946
00947
00948
00949
00950
00951
00952
00953
00954
00955
00956
00957
00958
00959
00960
00961
00962
00963
00964
00965
00966
00967
00968
00969
00970
00971
00972
00973
00974
00975
00976
00977
00978
00979
00980
00981
00982
00983
00984
00985
00986
00987
00988
00989
00990
00991
00992
00993
00994
00995
00996
00997
00998
00999
01000
01001
01002
01003
01004
01005
01006
01007
01008
01009
01010
01011
01012
01013
01014
01015
01016
01017
01018
01019
01020
01021
01022
01023
01024
01025
01026
01027
01028
01029
01030
01031
01032
01033
01034
01035
01036
01037
01038
01039
01040
01041
01042
01043
01044
01045
01046
01047
01048
01049
01050
01051
01052
01053
01054
01055
01056
01057
01058
01059
01060
01061
01062
01063
01064
01065
01066
01067
01068
01069
01070
01071
01072
01073
01074
01075
01076
01077
01078
01079
01080
01081
01082
01083
01084
01085
01086
01087
01088
01089
01090
01091
01092
01093
01094
01095
01096
01097
01098
01099
01100
01101
01102
01103
01104
01105
01106
01107
01108
01109
01110
01111
01112
01113
01114
01115
01116
01117
01118
01119
01120
01121
01122
01123
01124
01125
01126
01127
01128
01129
01130
01131
01132
01133
01134
01135
01136
01137
01138
01139
01140
01141
01142
01143
01144
01145
01146
01147
01148
01149
01150
01151
01152
01153
01154
01155
01156
01157
01158
01159
01160
01161
01162
01163
01164
01165
01166
01167
01168
01169
01170
01171
01172
01173
01174
01175
01176
01177
01178
01179
01180
01181
01182
01183
01184
01185
01186
01187
01188
01189
01190
01191
01192
01193
01194
01195
01196
01197
01198
01199
01200
01201
01202
01203
01204
01205
01206
01207
01208
01209
01210
01211
01212
01213
01214
01215
01216
01217
01218
01219
01220
01221
01222
01223
01224
01225
01226
01227
01228
01229
01230
01231
01232
01233
01234
01235
01236
01237
01238
01239
01240
01241
01242
01243
01244
01245
01246
01247
01248
01249
01250
01251
01252
01253
01254
01255
01256
01257
01258
01259
01260
01261
01262
01263
01264
01265
01266
01267
01268
01269
01270
01271
01272
01273
01274
01275
01276
01277
01278
01279
01280
01281
01282
01283
01284
01285
01286
01287
01288
01289
01290
01291
01292
01293
01294
01295
01296
01297
01298
01299
01300
01301
01302
01303
01304
01305
01306
01307
01308
01309
01310
01311
01312
01313
01314
01315
01316
01317
01318
01319
01320
01321
01322
01323
01324
01325
01326
01327
01328
01329
01330
01331
01332
01333
01334
01335
01336
01337
01338
01339
01340
01341
01342
01343
01344
01345
01346
01347
01348
01349
01350
01351
01352
01353
01354
01355
01356
01357
01358
01359
01360
01361
01362
01363
01364
01365
01366
01367
01368
01369
01370
01371
01372
01373
01374
01375
01376
01377
01378
01379
01380
01381
01382
01383
01384
01385
01386
01387
01388
01389
01390
01391
01392
01393
01394
01395
01396
01397
01398
01399
01400
01401
01402
01403
01404
01405
01406
01407
01408
01409
01410
01411
01412
01413
01414
01415
01416
01417
01418
01419
01420
01421
01422
01423
01424
01425
01426
01427
01428
01429
01430
01431
01432
01433
01434
01435
01436
01437
01438
01439
01440
01441
01442
01443
01444
01445
01446
01447
01448
01449
01450
01451
01452
01453
01454
01455
01456
01457
01458
01459
01460
01461
01462
01463
01464
01465
01466
01467
01468
01469
01470
01471
01472
01473
01474
01475
01476
01477
01478
01479
01480
01481
01482
01483
01484
01485
01486
01487
01488
01489
01490
01491
01492
01493
01494
01495
01496
01497
01498
01499
01500
01501
01502
01503
01504
01505
01506
01507
01508
01509
01510
01511
01512
01513
01514
01515
01516
01517
01518
01519
01520
01521
01522
01523
01524
01525
01526
01527
01528
01529
01530
01531
01532
01533
01534
01535
01536
01537
01538
01539
01540
01541
01542
01543
01544
01545
01546
01547
01548
01549
01550
01551
01552
01553
01554
01555
01556
01557
01558
01559
01560
01561
01562
01563
01564
01565
01566
01567
01568
01569
01570
01571
01572
01573
01574
01575
01576
01577
01578
01579
01580
01581
01582
01583
01584
01585
01586
01587
01588
01589
01590
01591
01592
01593
01594
01595
01596
01597
01598
01599
01600
01601
01602
01603
01604
01605
01606
01607
01608
01609
01610
01611
01612
01613
01614
01615
01616
01617
01618
01619
01620
01621
01622
01623
01624
01625
01626
01627
01628
01629
01630
01631
01632
01633
01634
01635
01636
01637
01638
01639
01640
01641
01642
01643
01644
01645
01646
01647
01648
01649
01650
01651
01652
01653
01654
01655
01656
01657
01658
01659
01660
01661
01662
01663
01664
01665
01666
01667
01668
01669
01670
01671
01672
01673
01674
01675
01676
01677
01678
01679
01680
01681
01682
01683
01684
01685
01686
01687
01688
01689
01690
01691
01692
01693
01694
01695
01696
01697
01698
01699
01700
01701
01702
01703
01704
01705
01706
01707
01708
01709
01710
01711
01712
01713
01714
01715
01716
01717
01718
01719
01720
01721
01722
01723
01724
01725
01726
01727
01728
01729
01730
01731
01732
01733
01734
01735
01736
01737
01738
01739
01740
01741
01742
01743
01744
01745
01746
01747
01748
01749
01750
01751
01752
01753
01754
01755
01756
01757
01758
01759
01760
01761
01762
01763
01764
01765
01766
01767
01768
01769
01770
01771
01772
01773
01774
01775
01776
01777
01778
01779
01780
01781
01782
01783
01784
01785
01786
01787
01788
01789
01790
01791
01792
01793
01794
01795
01796
01797
01798
01799
01800
01801
01802
01803
01804
01805
01806
01807
01808
01809
01810
01811
01812
01813
01814
01815
01816
01817
01818
01819
01820
01821
01822
01823
01824
01825
01826
01827
01828
01829
01830
01831
01832
01833
01834
01835
01836
01837
01838
01839
01840
01841
01842
01843
01844
01845
01846
01847
01848
01849
01850
01851
01852
01853
01854
0185
```

```

00236         }
00237     return FSM_Stop;
00238 }
00239
00240 if (I2C2STATbits.ACKSTAT==0) //Wenn ACK von Slave erhalten
00241 {
00242     static int count=0;
00243
00244     if (count < I2C_test_struct.num_write) //Noch Bytes zu senden
00245     {
00246
00247         I2C2TRN=I2C_test_struct.writebuf[count];
00248         count++;
00249         return FSM_Adresse_Write;
00250     }
00251
00252     else //Nichts mehr zu schicken
00253     {
00254         count=0;
00255         I2C2CONbits.RSEN=1; // Leitet Bedingungen für den Restart weiter
00256         return FSM_Repeated_Start;
00257     }
00258
00259 }
00260
00261 }
00262 return FSM_Adresse_Write;
00263
00264 } /* *FSM_Adresse_Write() */
00265
00271 void *FSM_Repeated_Start(void)
00272 {
00273     if (I2C2CONbits.RSEN==0) // Wenn der Restart erfolgreich war
00274     {
00275         I2C2TRN=(I2C_test_struct.address<<1) | 0b1;
00276         return FSM_Adresse_Read;
00277     }
00278     return FSM_Repeated_Start;
00279
00280 } /* *FSM_Repeated_Start() */
00281
00282
00294 void *FSM_Adresse_Read(void)
00295 {
00296     if(I2C2STATbits.TRSTAT==0) //Wenn erfolgreich übertragen
00297     {
00298         if (I2C2STATbits.ACKSTAT==1) //Wenn NACK von Slave erhalten
00299         {
00300             I2C2CONbits.PEN=1; // Leitet Stop-Bedigungen weiter
00301             *I2C_test_struct.status=Error; //Fehler bei Kommunikation
00302             if (I2C_test_struct.callback != NULL)
00303             {
00304                 I2C_test_struct.callback(I2C_test_struct.readbuf, I2C_test_struct.num_read,
00305                 I2C_test_struct.status, I2C_test_struct.ID);
00306             }
00307             return FSM_Stop;
00308         }
00309         if (I2C2STATbits.ACKSTAT==0) //Wenn ACK von Slave erhalten
00310         {
00311             if (I2C2CONbits.ACKEN==0) //Wenn Bit der ACK-Sequenz freigegeben
00312             {
00313                 static int count = 0;
00314
00315                 if (count < I2C_test_struct.num_read) //Noch Bytes zu empfangen
00316                 {
00317                     count++;
00318                     I2C2CONbits.RCEN=1; // Aktiviert den Empfangsmodus für I2C
00319                     return FSM_RECV_EN;
00320
00321
00322                 else //Nichts mehr zu empfangen
00323                 {
00324                     count = 0;
00325                     I2C2CONbits.PEN=1; // Leitet Stop-Bedigungen weiter
00326
00327                     *I2C_test_struct.status=Finished; //Anforderung abgearbeitet
00328                     //Callback Funktion aufrufen
00329                     if (I2C_test_struct.callback != NULL)
00330                     {
00331                         I2C_test_struct.callback(I2C_test_struct.readbuf, I2C_test_struct.num_read,
00332                         I2C_test_struct.status, I2C_test_struct.ID);
00333                     }
00334                     return FSM_Stop;
00335
00336             }
00337         }
00338     }
00339 }
```

```

00337         else
00338     {
00339         return FSM_Adresse_Read;
00340     }
00341 }
00342 }
00343 }
00344 }
00345 return FSM_Adresse_Read;
00346
00347 } /* *FSM_Adresse_Read() */
00348
00349 void *FSM_RECV_EN(void)
00350 {
00351     if (I2C2CONbits.RCEN==0) //Wenn der Empfangsmodus aktiviert wurde
00352     {
00353         static int count = 0;
00354         I2C_test_struct.readbuf[count]=I2C2RCV;
00355         count++;
00356
00357         if (count>=I2C_test_struct.num_read) //Wenn letztes Byte empfangen wurde
00358         {
00359             count=0;
00360             I2C2CONbits.ACKDT=1; //Sendet einen NACK während eines Acknowledge
00361         }
00362         else
00363         {
00364             I2C2CONbits.ACKDT=0; //Sendet einen ACK während eines Acknowledge
00365         }
00366
00367         I2C2CONbits.ACKEN=1; //Initiiert die Acknowledge-Sequenz
00368     }
00369     return FSM_Adresse_Read;
00370 }
00371
00372 return FSM_RECV_EN;
00373
00374 } /* *FSM_RECV_EN() */
00375
00376 void *FSM_Stop(void)
00377 {
00378     if(I2C2CONbits.PEN==0) //Wenn die Stop-Bedingungen weitergeleitet wurden
00379     {
00380         return FSM_Idle;
00381     }
00382     return FSM_Stop;
00383 }
00384
00385 } /* *FSM_Stop() */
00386
00387 void I2C_TempSens_Callback(uint8_t *readbuf, uint16_t num_read, i2c_status_t *status, int16_t ID)
00388 {
00389     if (ID==0) //Temperatur
00390     {
00391         if (*status==Finished)
00392         {
00393             double temp = readbuf[0] << 8 | readbuf[1];
00394             temp = temp / 256;
00395
00396             char str[32];
00397             sprintf(str,"Temperatur: %.1f Grad",temp);
00398             putsUART(str);
00399             char lf[2];
00400             sprintf(lf, "\n");
00401             putsUART(lf);
00402
00403             latest_temperatur=temp;
00404             *status=Pending;
00405         }
00406         if (*status==Error)
00407         {
00408             char str[32];
00409             sprintf(str,"Fehler Auslesen des Temp-Sensors!");
00410             putsUART(str);
00411             char lf[2];
00412             sprintf(lf, "\n");
00413             putsUART(lf);
00414         }
00415     }
00416 }
00417
00418 } /* *I2C_TempSens_Callback() */
00419
00420 void I2C_LightSens_Callback(uint8_t *readbuf, uint16_t num_read, i2c_status_t *status, int16_t ID)
00421 {
00422     if (ID==1)//Licht
00423     {
00424         if (*status==Finished)
00425
00426         if (*status==Pending)
00427         {
00428             if (*status==Error)
00429             {
00430                 if (*status==Pending)
00431                 {
00432                     if (*status==Error)
00433                     {
00434                         if (*status==Error)
00435                         {
00436                             if (*status==Error)
00437                             {
00438                                 if (*status==Error)
00439                                 {
00440                                     if (*status==Error)
00441                                     {
00442                                         if (*status==Error)
00443                                         {
00444                                             if (*status==Error)
00445                                             {
00446                                                 if (*status==Error)
00447                                                 {
00448                                                     if (*status==Error)
00449                                                     {
00450                                                         if (*status==Error)
00451                                                         {
00452
00453
00454
00455
00456
00457
00458
00459
00460
00461
00462
00463
00464
00465
00466
00467
00468
00469
00470
00471
00472
00473
00474
00475
00476
00477
00478
00479
00480
00481
00482
00483
00484
00485
00486
00487
00488
00489
00490
00491
00492
00493
00494
00495
00496
00497
00498
00499
00500
00501
00502
00503
00504
00505
00506
00507
00508
00509
00510
00511
00512
00513
00514
00515
00516
00517
00518
00519
00520
00521
00522
00523
00524
00525
00526
00527
00528
00529
00530
00531
00532
00533
00534
00535
00536
00537
00538
00539
00540
00541
00542
00543
00544
00545
00546
00547
00548
00549
00550
00551
00552
00553
00554
00555
00556
00557
00558
00559
00560
00561
00562
00563
00564
00565
00566
00567
00568
00569
00570
00571
00572
00573
00574
00575
00576
00577
00578
00579
00580
00581
00582
00583
00584
00585
00586
00587
00588
00589
00590
00591
00592
00593
00594
00595
00596
00597
00598
00599
00600
00601
00602
00603
00604
00605
00606
00607
00608
00609
00610
00611
00612
00613
00614
00615
00616
00617
00618
00619
00620
00621
00622
00623
00624
00625
00626
00627
00628
00629
00630
00631
00632
00633
00634
00635
00636
00637
00638
00639
00640
00641
00642
00643
00644
00645
00646
00647
00648
00649
00650
00651
00652
00653
00654
00655
00656
00657
00658
00659
00660
00661
00662
00663
00664
00665
00666
00667
00668
00669
00670
00671
00672
00673
00674
00675
00676
00677
00678
00679
00680
00681
00682
00683
00684
00685
00686
00687
00688
00689
00690
00691
00692
00693
00694
00695
00696
00697
00698
00699
00700
00701
00702
00703
00704
00705
00706
00707
00708
00709
00710
00711
00712
00713
00714
00715
00716
00717
00718
00719
00720
00721
00722
00723
00724
00725
00726
00727
00728
00729
00730
00731
00732
00733
00734
00735
00736
00737
00738
00739
00740
00741
00742
00743
00744
00745
00746
00747
00748
00749
00750
00751
00752
00753
00754
00755
00756
00757
00758
00759
00760
00761
00762
00763
00764
00765
00766
00767
00768
00769
00770
00771
00772
00773
00774
00775
00776
00777
00778
00779
00780
00781
00782
00783
00784
00785
00786
00787
00788
00789
00790
00791
00792
00793
00794
00795
00796
00797
00798
00799
00800
00801
00802
00803
00804
00805
00806
00807
00808
00809
00810
00811
00812
00813
00814
00815
00816
00817
00818
00819
00820
00821
00822
00823
00824
00825
00826
00827
00828
00829
00830
00831
00832
00833
00834
00835
00836
00837
00838
00839
00840
00841
00842
00843
00844
00845
00846
00847
00848
00849
00850
00851
00852
00853
00854
00855
00856
00857
00858
00859
00860
00861
00862
00863
00864
00865
00866
00867
00868
00869
00870
00871
00872
00873
00874
00875
00876
00877
00878
00879
00880
00881
00882
00883
00884
00885
00886
00887
00888
00889
00890
00891
00892
00893
00894
00895
00896
00897
00898
00899
00900
00901
00902
00903
00904
00905
00906
00907
00908
00909
00910
00911
00912
00913
00914
00915
00916
00917
00918
00919
00920
00921
00922
00923
00924
00925
00926
00927
00928
00929
00930
00931
00932
00933
00934
00935
00936
00937
00938
00939
00940
00941
00942
00943
00944
00945
00946
00947
00948
00949
00950
00951
00952
00953
00954
00955
00956
00957
00958
00959
00960
00961
00962
00963
00964
00965
00966
00967
00968
00969
00970
00971
00972
00973
00974
00975
00976
00977
00978
00979
00980
00981
00982
00983
00984
00985
00986
00987
00988
00989
00990
00991
00992
00993
00994
00995
00996
00997
00998
00999
01000
01001
01002
01003
01004
01005
01006
01007
01008
01009
01010
01011
01012
01013
01014
01015
01016
01017
01018
01019
01020
01021
01022
01023
01024
01025
01026
01027
01028
01029
01030
01031
01032
01033
01034
01035
01036
01037
01038
01039
01040
01041
01042
01043
01044
01045
01046
01047
01048
01049
01050
01051
01052
01053
01054
01055
01056
01057
01058
01059
01060
01061
01062
01063
01064
01065
01066
01067
01068
01069
01070
01071
01072
01073
01074
01075
01076
01077
01078
01079
01080
01081
01082
01083
01084
01085
01086
01087
01088
01089
01090
01091
01092
01093
01094
01095
01096
01097
01098
01099
01100
01101
01102
01103
01104
01105
01106
01107
01108
01109
01110
01111
01112
01113
01114
01115
01116
01117
01118
01119
01120
01121
01122
01123
01124
01125
01126
01127
01128
01129
01130
01131
01132
01133
01134
01135
01136
01137
01138
01139
01140
01141
01142
01143
01144
01145
01146
01147
01148
01149
01150
01151
01152
01153
01154
01155
01156
01157
01158
01159
01160
01161
01162
01163
01164
01165
01166
01167
01168
01169
01170
01171
01172
01173
01174
01175
01176
01177
01178
01179
01180
01181
01182
01183
01184
01185
01186
01187
01188
01189
01190
01191
01192
01193
01194
01195
01196
01197
01198
01199
01200
01201
01202
01203
01204
01205
01206
01207
01208
01209
01210
01211
01212
01213
01214
01215
01216
01217
01218
01219
01220
01221
01222
01223
01224
01225
01226
01227
01228
01229
01230
01231
01232
01233
01234
01235
01236
01237
01238
01239
01240
01241
01242
01243
01244
01245
01246
01247
01248
01249
01250
01251
01252
01253
01254
01255
01256
01257
01258
01259
01260
01261
01262
01263
01264
01265
01266
01267
01268
01269
01270
01271
01272
01273
01274
01275
01276
01277
01278
01279
01280
01281
01282
01283
01284
01285
01286
01287
01288
01289
01290
01291
01292
01293
01294
01295
01296
01297
01298
01299
01300
01301
01302
01303
01304
01305
01306
01307
01308
01309
01310
01311
01312
01313
01314
01315
01316
01317
01318
01319
01320
01321
01322
01323
01324
01325
01326
01327
01328
01329
01330
01331
01332
01333
01334
01335
01336
01337
01338
01339
01340
01341
01342
01343
01344
01345
01346
01347
01348
01349
01350
01351
01352
01353
01354
01355
01356
01357
01358
01359
01360
01361
01362
01363
01364
01365
01366
01367
01368
01369
01370
01371
01372
01373
01374
01375
01376
01377
01378
01379
01380
01381
01382
01383
01384
01385
01386
01387
01388
01389
01390
01391
01392
01393
01394
01395
01396
01397
01398
01399
01400
01401
01402
01403
01404
01405
01406
01407
01408
01409
01410
01411
01412
01413
01414
01415
01416
01417
01418
01419
01420
01421
01422
01423
01424
01425
01426
01427
01428
01429
01430
01431
01432
01433
01434
01435
01436
01437
01438
01439
01440
01441
01442
01443
01444
01445
01446
01447
01448
01449
01450
01451
01452
01453
01454
01455
01456
01457
01458
01459
01460
01461
01462
01463
01464
01465
01466
01467
01468
01469
01470
01471
01472
01473
01474
01475
01476
01477
01478
01479
01480
01481
01482
01483
01484
01485
01486
01487
01488
01489
01490
01491
01492
01493
01494
01495
01496
01497
01498
01499
01500
01501
01502
01503
01504
01505
01506
01507
01508
01509
01510
01511
01512
01513
01514
01515
01516
01517
01518
01519
01520
01521
01522
01523
01524
01525
01526
01527
01528
01529
01530
01531
01532
01533
01534
01535
01536
01537
01538
01539
01540
01541
01542
01543
01544
01545
01546
01547
01548
01549
01550
01551
01552
01553
01554
01555
01556
01557
01558
01559
01560
01561
01562
01563
01564
01565
01566
01567
01568
01569
01570
01571
01572
01573
01574
01575
01576
01577
01578
01579
01580
01581
01582
01583
01584
01585
01586
01587
01588
01589
01590
01591
01592
01593
01594
01595
01596
01597
01598
01599
01600
01601
01602
01603
01604
01605
01606
01607
01608
01609
01610
01611
01612
01613
01614
01615
01616
01617
01618
01619
01620
01621
01622
01623
01624
01625
01626
01627
01628
01629
01630
01631
01632
01633
01634
01635
01636
01637
01638
01639
01640
01641
01642
01643
01644
01645
01646
01647
01648
01649
01650
01651
01652
01653
01654
01655
01656
01657
01658
01659
01660
01661
01662
01663
01664
01665
01666
01667
01668
01669
01670
01671
01672
01673
01674
01675
01676
01677
01678
01679
01680
01681
01682
01683
01684
01685
01686
01687
01688
01689
01690
01691
01692
01693
01694
01695
01696
01697
01698
01699
01700
01701
01702
01703
01704
01705
01706
01707
01708
01709
01710
01711
01712
01713
01714
01715
01716
01717
01718
01719
01720
01721
01722
01723
01724
01725
01726
01727
01728
01729
01730
01731
01732
01733
01734
01735
01736
01737
01738
01739
01740
01741
01742
01743
01744
01745
01746
01747
01748
01749
01750
01751
01752
01753
01754
01755
01756
01757
01758
01759
01760
01761
01762
01763
01764
01765
01766
01767
01768
01769
01770
01771
01772
01773
01774
01775
01776
01777
01778
01779
01780
01781
01782
01783
01784
01785
01786
01787
01788
01789
01790
01791
01792
01793
01794
01795
01796
01797
01798
01799
01800
01801
01802
01803
01804
01805
01806
01807
01808
01809
01810
01811
01812
01813
01814
01815
01816
01817
01818
01819
01820
01821
01822
01823
01824
01825
01826
01827
01828
01829
01830
01831
01832
01833
01834
01835
01836
01837
01838
01839
01840
01841
01842
01843
01844
01845
01846
01847
01848
01849
01850
01851
01852
01853
01854
01855
01856
01857
01858
01859
01860
01861
01862
01863
01864
01865
01866
01867
01868
01869
01870
01871
01872
01873
01874
01875
01876
01877
01878
01879
01880
01881
01882
01883
01884
01885
01886
01887
01888
01889
01890
01891
01892
01893
01894
01895
01896
01897
01898
01899
01900
01901
01902
01903
01904
01905
01906
01907
01908
01909
01910
01911
01912
01913
01914
01915
01916
01917
01918
01919
01920
01921
01922
01923
01924
01925
01926
01927
01928
01929
01930
01931
01932
01933
01934
01935
01936
01937
01938
01939
01940
01941
01942
01943
01944
01945
01946
01947
01948
01949
01950
01951
01952
01953
01954
01955
01956
01957
01958
01959
01960
01961
01962
01963
01964
01965
01966
01967
01968
01969
01970
01971
01972
01973
01974
01975
01976
01977
01978
01979
01980
01981
01982
01983
01984
01985
01986
01987
01988
01989
01990
01991
01992
01993
01994
01995
01996
01997
01998
01999
02000
02001
02002
02003
02004
02005
02006
02007
02008
02009
02010
02011
02012
02013
02014
02015
02016
02017
02018
02019
02020
02021
02022
02023
02024
02025
02026
02027
02028
02029
02030
02031
02032
02033
02034
02035
02036
02037
02038
02039
02040
02041
02042
02043
02044
02045
02046
02047
02048
02049
02050
02051
02052
02053
02054
02055
02056
02057
02058
02059
02060
02061
02062
02063
02064
02065
02066
02067
02068
02069
02070
02071
02072
02073
02074
02075
02076
02077
02078
02079
02080
02081
02082
02083
02084
02085
02
```

```

00455     {
00456
00457     double light = readbuf[0] << 8 | readbuf[1];
00458     light = light / 1.2;
00459
00460     char str[16];
00461     sprintf(str,"Licht: %.1f lux",light);
00462     putsUART(str);
00463     char lf[2];
00464     sprintf(lf, "\n");
00465     putsUART(lf);
00466
00467     *status=Pending;
00468 }
00469
00470 if(*status==Error)
00471 {
00472     char str[32];
00473     sprintf(str,"Fehler Auslesen des Licht-Sensors!");
00474     putsUART(str);
00475     char lf[2];
00476     sprintf(lf, "\n");
00477     putsUART(lf);
00478 }
00479 }
00480 }/*I2C_LightSens_Callback()*/
00482

```

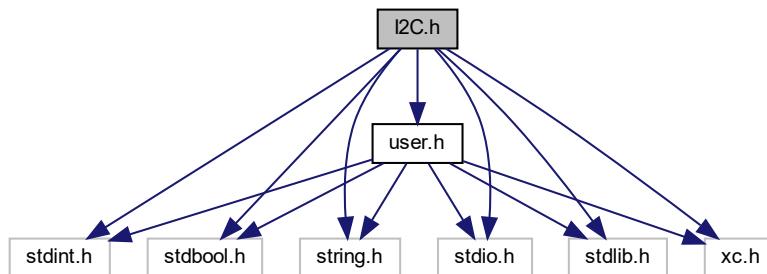
10.5 I2C.h-Dateireferenz

```

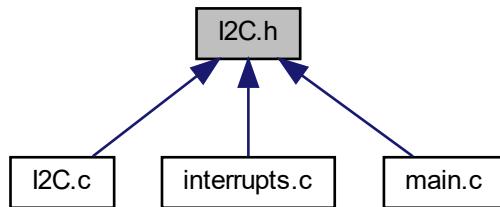
#include "user.h"
#include <stdint.h>
#include <stdbool.h>
#include <string.h>
#include <stdio.h>
#include <stdlib.h>
#include <xc.h>

```

Include-Abhängigkeitsdiagramm für I2C.h:



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



Datenstrukturen

- struct `I2C_struct`
Datenstruktur für die Kapselung aller benötigten Variablen, welche für ein character basiertes FIFO benötigt werden.
- 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(* I2C_Callback_t) (uint8_t *readbuf, uint16_t num_read, i2c_status_t *status, int16_t ID)`
- `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, I2C_Callback_t callback, int16_t ID)`
Übergibt angeforderte I2C-Anfrage an das FIFO und liefert den akutellen Status.
- `void I2C_TempSens_Callback (uint8_t *readbuf, uint16_t num_read, i2c_status_t *status, int16_t ID)`
Callback-Funktion zum Augeben der Temperaturwerte.
- `void I2C_LightSens_Callback (uint8_t *readbuf, uint16_t num_read, i2c_status_t *status, int16_t ID)`
Callback-Funktion zum Augeben der Lichtwerte.
- `void doI2C (void)`

- Wird jede ms in der Superloop ausgeführt und beinhaltet die FSM für die I2C-Kommunikation.*
- void [initI2C \(void\)](#)
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_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]
- i2c_status_t status_temperatur
- i2c_status_t status_licht
- double latest_temperatur
- I2C_struct I2C_test_struct
- Buffer_I2C_FSM FIFO_I2C

10.5.1 Makro-Dokumentation

10.5.1.1 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 24 der Datei [I2C.h](#).

10.5.1.2 I2C_SCL_TRIS

```
#define I2C_SCL_TRIS _TRISA2
```

Definiert in Zeile 26 der Datei [I2C.h](#).

10.5.1.3 I2C_SDA

```
#define I2C_SDA _RA3
```

Definiert in Zeile 25 der Datei [I2C.h](#).

10.5.1.4 I2C_SDA_TRIS

```
#define I2C_SDA_TRIS _TRISA3
```

Definiert in Zeile 27 der Datei [I2C.h](#).

10.5.2 Dokumentation der benutzerdefinierten Typen

10.5.2.1 I2C_Callback_t

```
typedef void(* I2C_Callback_t) (uint8_t *readbuf, uint16_t num_read, i2c_status_t *status,  
int16_t ID)
```

Definiert in Zeile 34 der Datei [I2C.h](#).

10.5.2.2 StateFunc

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

Definiert in Zeile 63 der Datei [I2C.h](#).

10.5.3 Dokumentation der Aufzählungstypen

10.5.3.1 i2c_status_t

```
enum i2c_status_t
```

Aufzählungswerte

Pending	
Finished	
Error	

Definiert in Zeile 33 der Datei [I2C.h](#).

10.5.4 Dokumentation der Funktionen

10.5.4.1 doI2C()

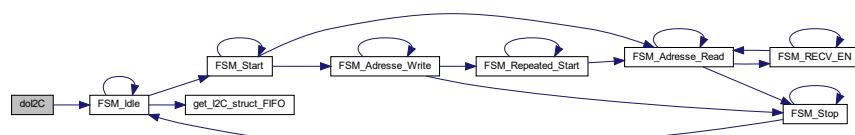
```
void doI2C (
    void )
```

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

Falls das FIFO neue Anfragen enthält wird die FSM getriggert.

Definiert in Zeile 117 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:



10.5.4.2 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,
    I2C_Callback_t callback,
    int16_t ID )
```

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

Parameter

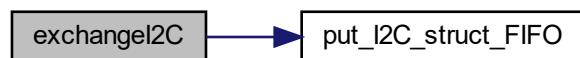
<i>address</i>	7 Bit Adresse des Slaves
<i>num_write</i>	Anzahl der zu sendenen Bytes, bei 0 keine Write Zugriff
<i>writebuf</i>	Zeiger auf zu schreibende Daten
<i>num_read</i>	Anzahl der zu lesenden Bytes, bei 0 keine Read Zugriff
<i>readbuf</i>	Zeiger auf Bereich, in welchem Daten abgespeichert werden sollen
<i>status</i>	Zeiger, um aktuellen Status zurückzugeben
<i>callback</i>	Zeiger auf eine Callback-Funktion, welche nach Abschluss der Anfrage aufgerufen werden soll
<i>ID</i>	ID zur Identifikation der Anfrage

Rückgabewerte

<i>1,Anforderung</i>	wurde angenommen, die FSM wird getriggert
<i>0,FSM</i>	ist beschäftigt, Anforderung kann nicht angeommen werden

Definiert in Zeile 96 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:



10.5.4.3 `FSM_Adresse_Read()`

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

Initiiert das Lesen der Daten des Slaves.

Parameter

<code>count</code>	Zaelervariable
--------------------	----------------

Rückgabe

`FSM_Stop`, sobald ein Fehler bei der Kommunikation auftritt, z.B kein ACK vom Slave. Es wird die Callback Funktion aufgerufen

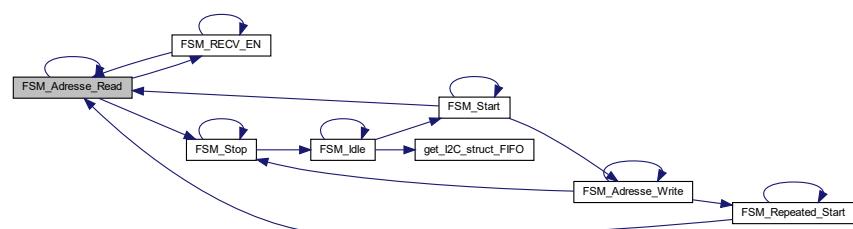
`FSM_RECV_EN`, sobald der Empfangsmodus für I2C aktiviert wurde

`FSM_Stop`, sobald die Stop-Bedigungen an die Pins SDAx und SCLx weitergeleitet wurden. Es wird die Callback Funktion aufgerufen

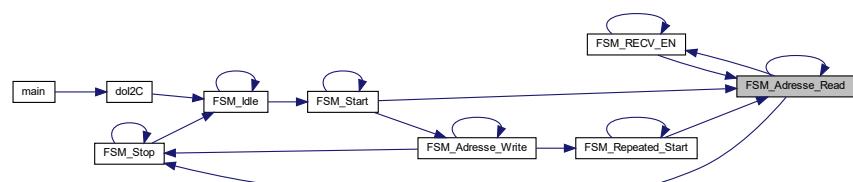
`FSM_Adresse_Read`, wenn kein ACK vom Slave erhalten oder das Bit der ACK-Sequenz nicht freigegeben ist

Definiert in Zeile 294 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:



10.5.4.4 `FSM_Adresse_Write()`

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

Schreibt die zu übertragende Daten in das Tranceive-Register.

Parameter

<code>count</code>	Zaelervariable
--------------------	----------------

Rückgabe

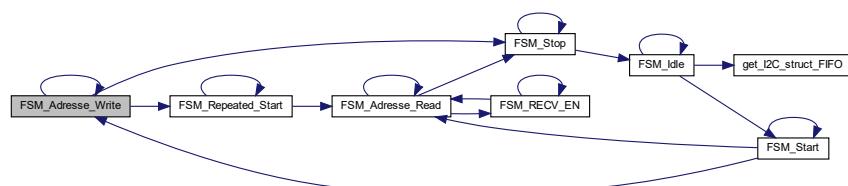
`FSM_Stop`, sobald ein Fehler bei der Kommunikation auftritt, z.B kein ACK vom Slave. Es wird die Callback Funktion aufgerufen

`FSM_Adresse_Write`, sobald keine Bytes mehr zu senden gibt

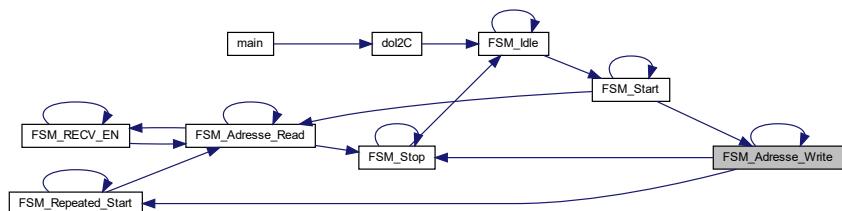
`FSM_Repeated_Start`, sobald die Bedingungen für den wiederholten Start an die Pins SDAx und SClx weitergeleitet wurde.

Definiert in Zeile [225](#) 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:



10.5.4.5 `FSM_Idle()`

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

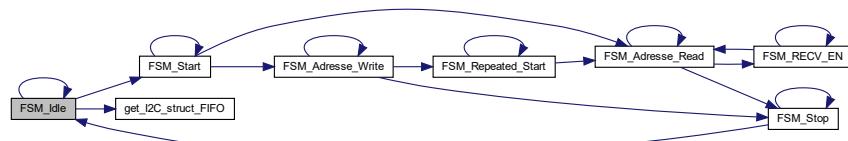
Kopiert die Anfrage aus dem FIFO und leitet Start-Sequenz ein.

Rückgabe

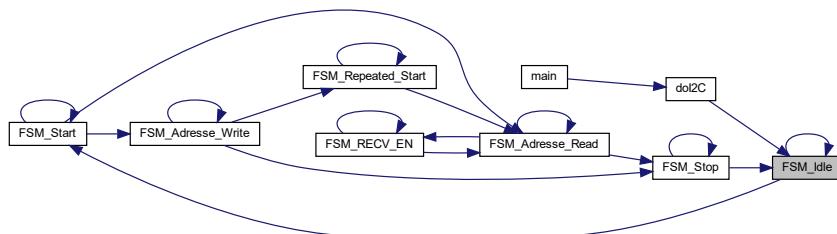
FSM_Start, sobald die Startbedingungen an die Pins SDAx und SCLx weitergeleitet worden sind

Definiert in Zeile 173 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:



10.5.4.6 `FSM_RECV_EN()`

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

Auslesen des Receive Registers und Bestätigung mit ACK bzw.

NACK

Parameter

<code>count</code>	Zaelervariable
--------------------	----------------

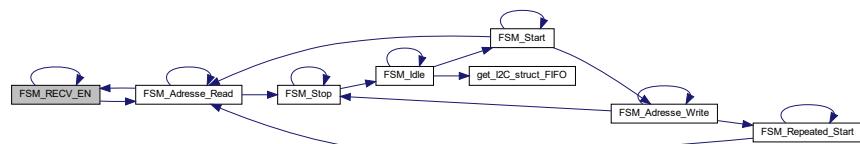
Rückgabe

FSM_Adresse_Read, sobald die Acknowledge-Sequenz an den Pins SDAx und SCLx initiiert wurde und das ACKDT Datenbit übertragen wurde

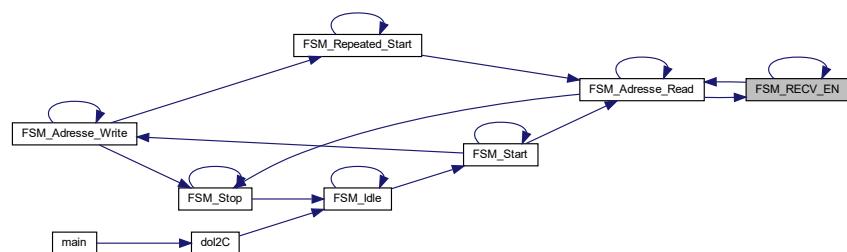
FSM_RECV_EN, Wenn die Empfangssequenz nicht ausgeführt wurde

Definiert in Zeile 356 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:



10.5.4.7 `FSM_Repeated_Start()`

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

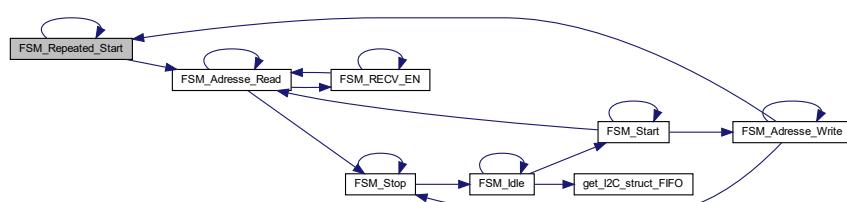
Leitet einen Repeated Start ein und beschreibt das Tranceive Register mit der Adresse.

Rückgabe

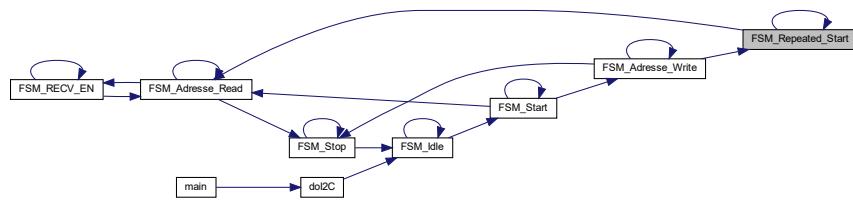
`FSM_Adresse_Read`, sobald es einen Restart gibt

Definiert in Zeile 271 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:



10.5.4.8 FSM_Start()

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

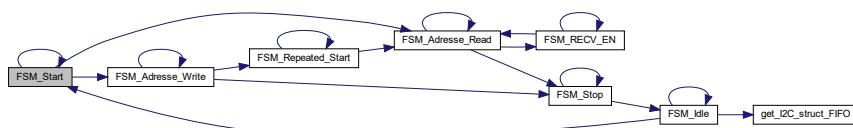
Beschreibt das Trancieve-Register mit der Adresse.

Rückgabe

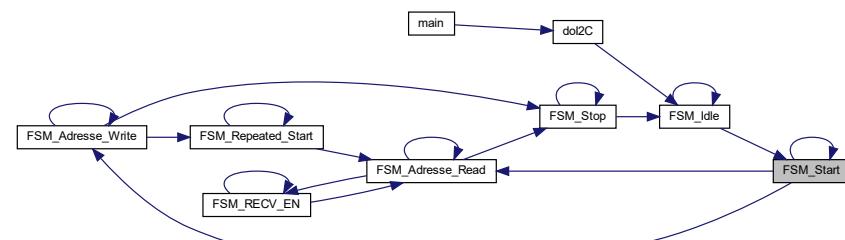
FSM_Adresse_Write, sobald geschrieben werden kann @retrun FSM_Adresse_Read, sobald gelesen werden kann

Definiert in Zeile 192 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:



10.5.4.9 FSM_Stop()

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

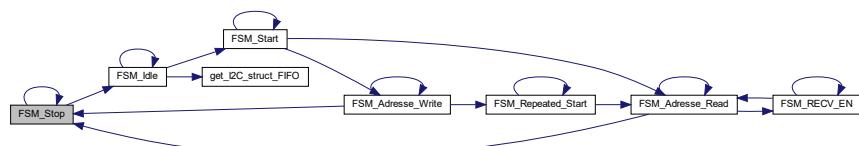
Überprüfung auf Abschluss der Stop-Sequenz und Rückkehr in den Idle-State.

Rückgabe

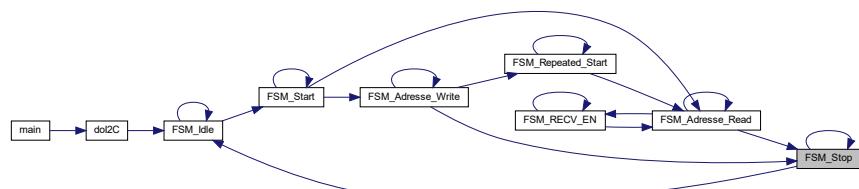
FSM_Idle, wenn die Stop-Bedingungen erfolgreich an den Pins SDAx und SCLx weitergeleitet wurden
 FSM_Stop, wenn keine Stop-Bedingungen weitergeleitet wurden

Definiert in Zeile 388 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:



10.5.4.10 I2C_LightSens_Callback()

```
void I2C_LightSens_Callback (
    uint8_t * readbuf,
    uint16_t num_read,
    i2c_status_t * status,
    int16_t ID )
```

Callback-Funktion zum Augeben der Lichtwerte.

Parameter

<i>readbuf</i>	Zeiger auf die ausgelesenen Daten
<i>num_read</i>	Anzahl der ausgelesenen Bytes
<i>status</i>	Status mit welchem die FSM die Callback-Funktion aufgerufen hat. Im Fall Error wird eine Fehlermeldung ausgegeben. Im Fall Finished werden die Daten interpretiert und ausgegeben.
<i>ID</i>	

Definiert in Zeile 450 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:



10.5.4.11 I2C_TempSens_Callback()

```

void I2C_TempSens_Callback (
    uint8_t * readbuf,
    uint16_t num_read,
    i2c_status_t * status,
    int16_t ID )
  
```

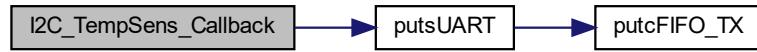
Callback-Funktion zum Augeben der Temperaturwerte.

Parameter

<i>readbuf</i>	Zeiger auf die ausgelesenen Daten
<i>num_read</i>	Anzahl der ausgelesenen Bytes
<i>status</i>	Status mit welchem die FSM die Callback-Funktion aufgerufen hat. Im Fall Error wird eine Fehlermeldung ausgegeben. Im Fall Finished werden die Daten interpretiert und ausgegeben.
<i>ID</i>	

Definiert in Zeile 408 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:



10.5.4.12 initI2C()

```
void initI2C ( void )
```

Initialisiert die I2C-Kommunikation.

Einschalten I2C mit eigenem Workaround, I2C Peripheriemodul kann hier leider nicht verwendet werden. Pins wie einen Open-Kollektor-Treiber verwenden, d.h. 1 - hochohmig, 0 wird getrieben

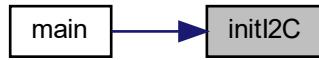
Aktiviert das I2C Modul und konfiguriert die Pins SDAx und SCLx als serielle PORT-Pins.

Definiert in Zeile 126 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:



10.5.5 Variablen-Dokumentation

10.5.5.1 FIFO_I2C

`Buffer_I2C_FSM FIFO_I2C [extern]`

Definiert in Zeile 18 der Datei [I2C.c](#).

10.5.5.2 I2C_test_struct

`I2C_struct I2C_test_struct [extern]`

Definiert in Zeile 16 der Datei [I2C.c](#).

10.5.5.3 latest_temperatur

`double latest_temperatur [extern]`

Definiert in Zeile 46 der Datei [main.c](#).

10.5.5.4 read_data_buffer_light

`uint8_t read_data_buffer_light[2] [extern]`

Definiert in Zeile 44 der Datei [main.c](#).

10.5.5.5 `read_data_buffer_temp`

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

Definiert in Zeile 43 der Datei [main.c](#).

10.5.5.6 `status_licht`

```
i2c_status_t status_licht [extern]
```

Definiert in Zeile 39 der Datei [main.c](#).

10.5.5.7 `status_temperatur`

```
i2c_status_t status_temperatur [extern]
```

Definiert in Zeile 38 der Datei [main.c](#).

10.5.5.8 `write_data_buffer_light`

```
uint8_t write_data_buffer_light [extern]
```

Definiert in Zeile 42 der Datei [main.c](#).

10.5.5.9 `write_data_buffer_temp`

```
uint8_t write_data_buffer_temp [extern]
```

Definiert in Zeile 41 der Datei [main.c](#).

10.6 I2C.h

[gehe zur Dokumentation dieser Datei](#)

```

00001
00006
00007 /* Files to Include */ 
00008
00009 #include "user.h"      /* Benutzer - Funktion/Parameter */
00010 // #include "UART.h"   /* Enthält uint16_t-Definition */
00011 #include <stdint.h>     /* Enthält eine Wahr/Falsch-Definition */
00012 #include <stdbool.h>    /* Enthält Zeichenketten */
00013 #include <string.h>     /* Enthält Ein - und Ausgabefunktionen */
00014 #include <stdio.h>      /* Enthält Hilfsfunktionen */
00015 #include <stdlib.h>
00016 #include <stdlib.h>
00017
00018 #include <xc.h>        /* Jede Prozessordatei ist geschützt. */
00019
00020
00021 /* Konstanten */ 
00022
00023 #define I2C_SCL      _RA2
00024 #define I2C_SDA      _RA3
00025 #define I2C_SCL_TRIS _TRISA2
00026 #define I2C_SDA_TRIS _TRISA3
00027
00028
00029
00030 /* Typedef */ 
00031
00032
00033 typedef enum {Pending, Finished, Error} i2c_status_t;
00034 typedef void (*I2C_Callback_t)(uint8_t *readbuf, uint16_t num_read, i2c_status_t *status, int16_t ID);
00035
00036
00037
00038
00039
00040
00041
00042
00043
00044
00045
00046
00047
00048
00049
00050
00051
00052
00053
00054
00055
00056
00057
00058
00059
00060
00061
00062
00063
00064
00065
00066
00067 /* Global Variable Declaration */ 
00068
00069
00070 // In Header nur Prototypen, genauso auch mit den Variablen. Variale sollte in c Datei deklariert
00071 // sein. In Header nur mit extern.
00072 extern uint8_t write_data_buffer_temp;
00073 extern uint8_t write_data_buffer_light;
00074 extern uint8_t read_data_buffer_temp[2];
00075 extern uint8_t read_data_buffer_light[2];
00076
00077
00078
00079
00080
00081
00082
00083
00084
00085
00086 /* Prototypen */ 
00087
00088 int16_t exchangeI2C(uint8_t address, uint16_t num_write, uint8_t *writebuf, uint16_t num_read, uint8_t
00089 *readbuf, i2c_status_t *status, I2C_Callback_t callback, int16_t ID);
00090
00091 void I2C_TempSens_Callback(uint8_t *readbuf, uint16_t num_read, i2c_status_t *status, int16_t ID);
00092 void I2C_LightSens_Callback(uint8_t *readbuf, uint16_t num_read, i2c_status_t *status, int16_t ID);
00093
00094 void doI2C(void);
00095
00096 void initI2C(void);

```

```

00097
00098 void *FSM_Idle(void);
00099 void *FSM_Start(void);
00100 void *FSM_Adresse_Read(void);
00101 void *FSM_Adresse_Write(void);
00102 void *FSM_Repeated_Start(void);
00103 void *FSM_RECV_EN(void);
00104 void *FSM_Stop(void);

```

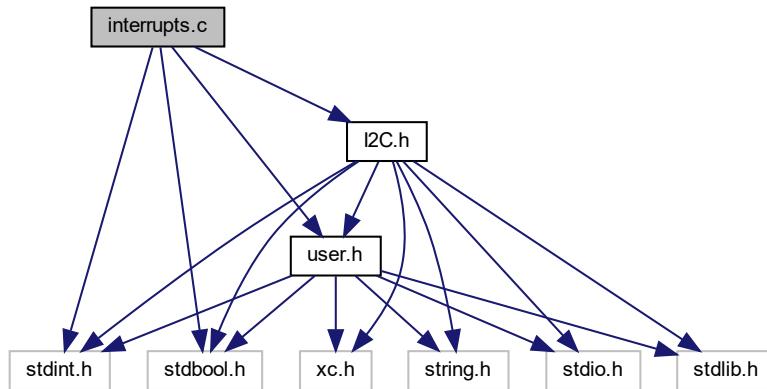
10.7 interrupts.c-Dateireferenz

```

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

```

Include-Abhängigkeitsdiagramm für interrupts.c:



Funktionen

- void `_T1Interrupt (void)`
Interrupt des Timer1, welcher jede Sekunde ausgeführt wird.

10.7.1 Dokumentation der Funktionen

10.7.1.1 `_T1Interrupt()`

```

void _T1Interrupt (
    void )

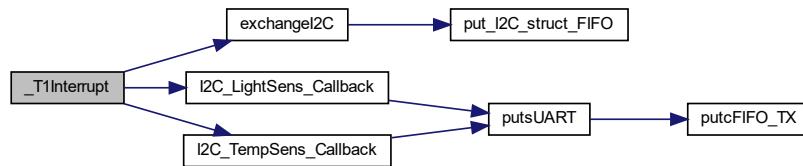
```

Interrupt des Timer1, welcher jede Sekunde ausgeführt wird.

Damit wird jede Sekunde eine I2C-Anfrage für den Licht- und den Temperatursensor gestellt. Als Erweiterung wird nun ein Zeiger auf die auszuführende Callback-Funktion sowie eine ID bei den Anfragen übergeben.

Definiert in Zeile 34 der Datei [interrupts.c](#).

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



10.8 interrupts.c

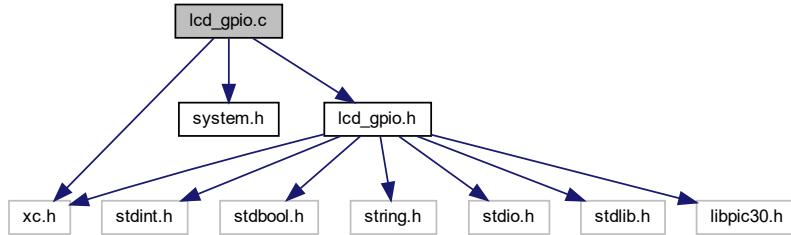
[gehe zur Dokumentation dieser Datei](#)

```

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 <p33Exxxxx.h>
00011     #elif defined(__dsPIC33F__)
00012         #include <p33Fxxxx.h>
00013     #endif
00014 #endif
00015
00016 #include <stdint.h>          /* Enthält uint16_t-Definition
00017 #include <stdbool.h>          /* Enthält eine Wahr/Falsch-Definition
00018
00019 #include "user.h"            /* Benutzer - Funktion/Parameter
00020
00021 /* Beinhaltet Konstanten, Typdefs, globale Variablen und Prototypen für main */
00022 #include "I2C.h"
00023
00024 /* Interrupt Routines
00026
00034 void __attribute__((__interrupt__, no_auto_psv)) _T1Interrupt(void)
00035 {
00036     _T1IF = 0; //Clear Timer1 interrupt flag
00037     static int count=0;
00038
00039     if (count>=SENSOR_TIME-1)
00040     {
00041         count=0;
00042         //Anfrage Temperatur-Sensor
00043         exchangeI2C(0b1001000, 1, &write_data_buffer_temp, 2, read_data_buffer_temp,
00044                     &status_temperatur, &I2C_TempSens_Callback, 0);
00045         //Anfrage Licht-Sensor
00046         exchangeI2C(0b0100011, 1, &write_data_buffer_light, 2, read_data_buffer_light, &status_licht,
00047                     &I2C_LightSens_Callback, 1);
00048     }
00049     else
00050     {
00051         count++;
00052     }
00053 }
```

10.9 lcd_gpio.c-Dateireferenz

```
#include <xc.h>
#include "system.h"
#include "lcd_gpio.h"
Include-Abhängigkeitsdiagramm für lcd_gpio.c:
```



Funktionen

- void [lcd_init \(\)](#)
Initialisierung des LCDs.
- void [lcd_write_data \(uint8_t data\)](#)
Schreiben von Daten auf den Speicher des LCDs.
- void [lcd_clear \(void\)](#)
Sendet den Befehl, das LCD zurückzusetzen und den Inhalt zu löschen.
- void [writeStrLCD \(const char *str\)](#)
Gibt einen String an der aktuellen Position im Display aus.
- void [lcd_set_pos \(int line, int pos\)](#)
Setzt die Position des Cursors.
- uint8_t [lcd_get_status \(void\)](#)
Liest den aktuellen Status des LCDs aus.
- void [waitForBusyLCD \(void\)](#)
Ruft [lcd_get_status\(\)](#) zyklisch auf, bis Busy Flag nicht mehr gesetzt ist.

10.9.1 Dokumentation der Funktionen

10.9.1.1 lcd_clear()

```
void lcd_clear (
    void )
```

Sendet den Befehl, das LCD zurückzusetzen und den Inhalt zu löschen.

Vor dem Zugriff wird Funktion [waitForBusyLCD\(\)](#) solange blockieren, bis Busy Flag nicht gesetzt.

Definiert in Zeile 135 der Datei [lcd_gpio.c](#).

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



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



10.9.1.2 lcd_get_status()

```
uint8_t lcd_get_status (
    void )
```

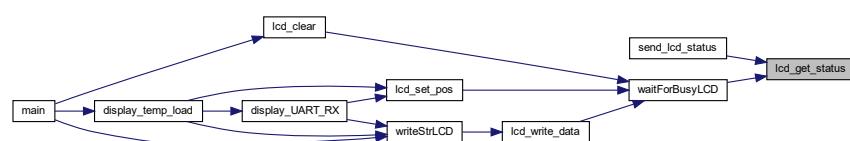
Liest den akutellen Status des LCDs aus.

Rückgabe

8bit breiter Status, Bit 7 ist Busy Flag

Definiert in Zeile 230 der Datei [lcd_gpio.c](#).

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



10.9.1.3 lcd_init()

```
void lcd_init (
    void  )
```

Initialisierung des LCDs.

Drei Function Sets und anschließend die Konfiguration des LCDs. Diese beinhaltet das Funktion Set, Display off, Display clear, Entry mode und Display on.

Definiert in Zeile 22 der Datei [lcd_gpio.c](#).

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



10.9.1.4 lcd_set_pos()

```
void lcd_set_pos (
    int line,
    int pos )
```

Setzt die Position des Cursors.

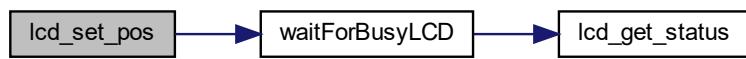
Vor dem Zugriff wird Funktion [waitForBusyLCD\(\)](#) solange blockieren, bis Busy Flag nicht gesetzt.

Parameter

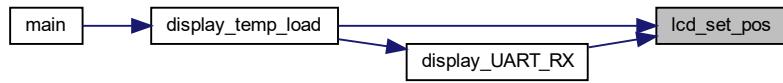
<i>line</i>	Linie, entweder 1 oder 2
<i>pos</i>	Position

Definiert in Zeile 175 der Datei [lcd_gpio.c](#).

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



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



10.9.1.5 lcd_write_data()

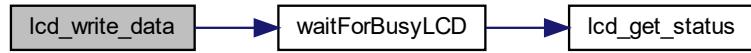
```
void lcd_write_data (
    uint8_t data )
```

Schreiben von Daten auf den Speicher des LCDs.

Vor dem Zugriff wird Funktion `waitForBusyLCD()` solange blockieren, bis Busy Flag nicht gesetzt.

Definiert in Zeile 114 der Datei `lcd_gpio.c`.

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



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



10.9.1.6 waitForBusyLCD()

```
void waitForBusyLCD (
    void )
```

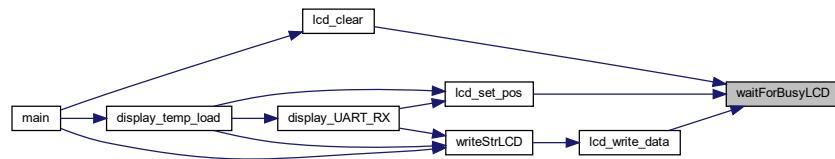
Ruft [lcd_get_status\(\)](#) zyklisch auf, bis Busy Flag nicht mehr gesetzt ist.

Definiert in Zeile [251](#) der Datei [lcd_gpio.c](#).

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



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



10.9.1.7 writeStrLCD()

```
void writeStrLCD (
    const char * str )
```

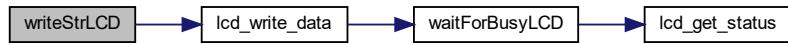
Gibt einen String an der aktuellen Position im Display aus.

Parameter

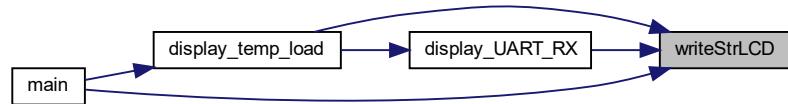
<code>str</code>	Zeichenkette
------------------	--------------

Definiert in Zeile [157](#) der Datei [lcd_gpio.c](#).

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



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



10.10 lcd_gpio.c

[gehe zur Dokumentation dieser Datei](#)

```

00001
00002 /* Files to Include
00003
00005 //TODO: Logic Analyzer Doku mit Stuersignalen, Bit7 +...
00006
00007 #include <xc.h>           /* Jede Prozessordatei ist geschützt. */
00008
00009 #include "system.h"         /* System - Funktion/Parameter
00010
00011 #include "lcd_gpio.h"       /* Behinhaltet Konstanten und Prototypen
00012
00013
00014 /* Funktionen
00015
00022 void lcd_init()
00023 {
00024     //Alle Signale als Ausgänge
00025     _TRISB15 = 0;           //RS
00026     _TRISD5 = 0;           //R_W
00027     _TRISD4 = 0;           //Enable
00028     TRISE &= 0xFF00;        //Datenbus als Ausgang
00029
00030     //Alle Ausgänge als Digital
00031     _ANSB15 = 0;
00032     ANSELE &= 0xFF00;
00033
00034     LCD_ENABLE = 0;         //LCD Aktivierungssignal
00035     LCD_RS = 0;            //LCD Registerauswahlsignal
00036     LCD_R_W = 0;           //LCD Daten Lesen oder Schreiben
00037
00038
00039     //Function Set 1
00040     delay_ms(40);          //40 ms warten
00041     LCD_DATA(LCD_CMD_INIT);
00042     LCD_ENABLE = 1;
00043     __delay_cycles(33);      //660 ns warten
00044     LCD_ENABLE = 0;
00045
00046
00047     //Function Set 2
00048     __delay_us(4100);        //4.1 ms warten
00049     LCD_DATA(LCD_CMD_INIT);
00050     LCD_ENABLE = 1;
00051     __delay_cycles(33);      //660 ns warten
00052     LCD_ENABLE = 0;
  
```

```
00053
00054
00055 //Function Set 3
00056     __delay_us(100);           //100 us warten
00057 LCD_DATA(LCD_CMD_INIT);
00058 LCD_ENABLE = 1;
00059 __delay_cycles(33);        //660 ns warten
00060 LCD_ENABLE = 0;
00061 __delay_us(38);           //LCD_CMD_INIT benötigt 38 us zum Ausführen
00062
00063
00064
00065 /*LCD_FUNCTION_SET stellt die Bewegungsrichtig von Cursor und Display ein.*/
00066 LCD_DATA(LCD_FUNCTION_SET);
00067 LCD_ENABLE = 1;
00068 __delay_cycles(33);        //660 ns warten
00069 LCD_ENABLE = 0;
00070 __delay_us(38);           //LCD_FUNCTION_SET benötigt 38 us zum Ausführen
00071
00072
00073
00074
00075 /*LCD_DISPLAY_OFF schaltet das Display aus.*/
00076 LCD_DATA(LCD_DISPLAY_OFF);
00077 LCD_ENABLE = 1;
00078 __delay_cycles(33);        //660 ns warten
00079 LCD_ENABLE = 0;
00080 __delay_us(38);           //LCD_DISPLAY_OFF benötigt 38 us zum Ausführen
00081
00082
00083 /*LCD_DISPLAY_CLEAR löscht die Anzeigedaten.*/
00084 LCD_DATA(LCD_DISPLAY_CLEAR);
00085 LCD_ENABLE = 1;
00086 __delay_cycles(33);        //660 ns warten
00087 LCD_ENABLE = 0;
00088 __delay_us(1520);          //LCD_DISPLAY_CLEAR benötigt 1.52 ms zum Ausführen
00089
00090
00091 /*LCD_ENTRY_MODE stellt den Eingabemodus ein. Der Cursor bewegt sich dabei
00092 *nach rechts.*/
00093 LCD_DATA(LCD_ENTRY_MODE);
00094 LCD_ENABLE = 1;
00095 __delay_cycles(33);        //660 ns warten
00096 LCD_ENABLE = 0;
00097 __delay_us(38);           //LCD_ENTRY_MODE benötigt 38 us zum Ausführen
00098
00099
00100 /*LCD_DISPLAY_ON schaltet das Display ein.*/
00101 LCD_DATA(LCD_DISPLAY_ON);
00102 LCD_ENABLE = 1;
00103 __delay_cycles(33);        //660 ns warten
00104 LCD_ENABLE = 0;
00105 __delay_us(38);           //LCD_DISPLAY_ON benötigt 38 us zum Ausführen
00106
00107 }/*lcd_init()*/
00108
00109
00110 void lcd_write_data(uint8_t data)
00111 {
00112     waitForBusyLCD();
00113     TRISE &= 0xFF00;           //Datenbus als Ausgang
00114
00115     LCD_RS = 1;
00116     LCD_R_W = 0;
00117     __delay_cycles(4);        //80 ns warten
00118     LCD_DATA(data);
00119     LCD_ENABLE = 1;
00120     __delay_cycles(33);        //660 ns warten
00121     LCD_ENABLE = 0;
00122
00123
00124
00125
00126
00127 }/*lcd_write_data()*/
00128
00129
00130 void lcd_clear(void)
00131 {
00132     waitForBusyLCD();
00133     TRISE &= 0xFF00;           //Datenbus als Ausgang
00134
00135     LCD_RS = 0;
00136     LCD_R_W = 0;
00137     __delay_cycles(4);        //80 ns warten
00138
00139
00140
00141
00142
00143
00144 /*LCD_DISPLAY_CLEAR löscht die Anzeigedaten.*/
00145 LCD_DATA(LCD_DISPLAY_CLEAR);
00146 LCD_ENABLE = 1;
00147 __delay_cycles(33);        //660 ns warten
00148 LCD_ENABLE = 0;
00149
00150 }/*lcd_clear()*/
```

```
00151
00152
00153 void writeStrLCD(const char* str)
00154 {
00155     uint8_t i = 0;
00156
00157     while (str[i]!=0)          //Solange es etwas zu schreiben gibt.
00158     {
00159         lcd_write_data((str[i]));
00160         i++;
00161     }
00162
00163 }/*writeStrLCD()*/
00164
00165 void lcd_set_pos(int line, int pos)
00166 {
00167     waitForBusyLCD();
00168     TRISE &= 0xFF00;           //Datenbus als Ausgang
00169
00170     //Position auf 0,0 setzen
00171     LCD_RS = 0;
00172     LCD_R_W = 0;
00173     __delay_cycles(4);       //80 ns warten
00174
00175     /* LCD_DISPLAY_HOME setzt Cursor und Anzeige an ihren ursprünglichen Zustand
00176      * zurück.*/
00177     LCD_DATA(LCD_DISPLAY_HOME);
00178     LCD_ENABLE = 1;
00179     __delay_cycles(33);      //660 ns warten
00180     LCD_ENABLE = 0;
00181
00182
00183     int i = 0;
00184     int to_shift = 0;
00185
00186     if (line == 1)           //Erste Linie
00187     {
00188         to_shift = pos;
00189     }
00190
00191     else                     //Zweite Linie
00192     {
00193         to_shift = pos + 40;
00194     }
00195
00196     for(i = 0; i < to_shift - 1; i++)
00197     {
00198         waitForBusyLCD();
00199         TRISE &= 0xFF00;           //Datenbus als Ausgang
00200
00201         LCD_RS = 0;
00202         LCD_R_W = 0;
00203         __delay_cycles(4);       //80 ns warten
00204
00205         /*CURSOR_OR_DISPLAY schiebt Cursor und Anzeige nach rechts (der Cursor
00206          * bewegt sich entsprechend der Anzeige).*/
00207         LCD_DATA(CURSOR_OR_DISPLAY);
00208         LCD_ENABLE = 1;
00209         __delay_cycles(33);      //660 ns warten
00210         LCD_ENABLE = 0;
00211     }
00212
00213 }/*lcd_set_pos()*/
00214
00215
00216 uint8_t lcd_get_status(void)
00217 {
00218     uint8_t received_data;
00219     TRISE = TRISE | 0x00FF; //Datenbus als Eingang
00220
00221     LCD_RS = 0;
00222     LCD_R_W = 1;
00223     __delay_cycles(4);       //80 ns warten
00224     LCD_ENABLE = 1;
00225     __delay_cycles(20);      //min 360 ns warten -> 400ns
00226     received_data = PORTE;
00227     LCD_ENABLE = 0;
00228
00229     return received_data;
00230
00231 }/*lcd_get_status()*/
00232
00233 void waitForBusyLCD(void)
00234 {
00235     /* READ_BUSY_FLAG definiert den IC und gibt an das dieser in Betrieb ist.
00236      * Interne Operation ist im Gange es soll gewartet werden.*/
00237 }
```

```

00255     while ((lcd_get_status() & READ_BUSY_FLAG))
00256     {
00257
00258     }
00259     return;
00260 }/*waitForBusyLCD ()*/

```

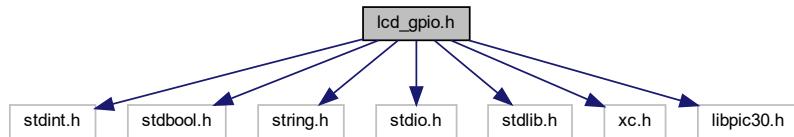
10.11 lcd_gpio.h-Dateireferenz

```

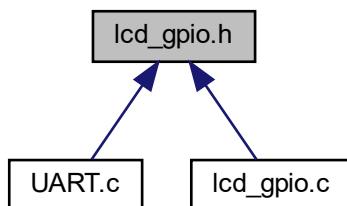
#include <stdint.h>
#include <stdbool.h>
#include <string.h>
#include <stdio.h>
#include <stdlib.h>
#include <xc.h>
#include "libpic30.h"

```

Include-Abhängigkeitsdiagramm für lcd_gpio.h:



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



Makrodefinitionen

- #define **_delay_cycles**(cycles) asm volatile ("repeat %%0 \n nop" : : "i" (cycles-2))
- #define **LCD_DATA**(d) LATE = (LATE & 0xFF00) | d
- #define **LCD_RS _RB15** /*LCD Registerauswahlsignal*/
- #define **LCD_R_W _RD5** /*LCD Daten Lesen oder Schreiben*/
- #define **LCD_ENABLE _RD4** /*LCD Aktivierungssignal*/
- #define **LCD_CMD_INIT** 0b0000000000110000 /*LCD wird Initialisiert*/

- #define **LCD_FUNCTION_SET** 0b00000000000111000
- #define **LCD_DISPLAY_OFF** 0b0000000000001000
- #define **LCD_DISPLAY_CLEAR** 0b00000000000000001
- #define **LCD_DISPLAY_HOME** 0b0000000000000000010
- #define **LCD_ENTRY_MODE** 0b000000000000000110
- #define **LCD_DISPLAY_ON** 0b0000000000000001110
- #define **READ_BUSY_FLAG** 0b0000000010000000
- #define **CURSOR_OR_DISPLAY** 0b00000000000010100

Funktionen

- void **lcd_init** (void)
Initialisierung des LCDs.
- void **lcd_write_data** (uint8_t **data**)
Schreiben von Daten auf den Speicher des LCDs.
- void **writeStrLCD** (const char *str)
Gibt einen String an der aktuellen Position im Display aus.
- void **lcd_clear** (void)
Sendet den Befehl, das LCD zurückzusetzen und den Inhalt zu löschen.
- void **lcd_set_pos** (int line, int pos)
Setzt die Position des Cursors.
- void **waitForBusyLCD** (void)
*Ruft **lcd_get_status()** zyklisch auf, bis Busy Flag nicht mehr gesetzt ist.*
- uint8_t **lcd_get_status** (void)
Liest den aktuellen Status des LCDs aus.

10.11.1 Makro-Dokumentation

10.11.1.1 __delay_cycles

```
#define __delay_cycles(
    cycles ) asm volatile ("repeat #%0 \n nop" : : "i" (cycles-2))
```

Definiert in Zeile 21 der Datei [lcd_gpio.h](#).

10.11.1.2 CURSOR_OR_DISPLAY

```
#define CURSOR_OR_DISPLAY 0b00000000000010100
```

Definiert in Zeile 61 der Datei [lcd_gpio.h](#).

10.11.1.3 LCD_CMD_INIT

```
#define LCD_CMD_INIT 0b0000000000110000 /*LCD wird Initialisiert*/
```

Definiert in Zeile 28 der Datei [lcd_gpio.h](#).

10.11.1.4 LCD_DATA

```
#define LCD_DATA( d ) LATE = (LATE & 0xFF00) | d
```

Definiert in Zeile 23 der Datei [lcd_gpio.h](#).

10.11.1.5 LCD_DISPLAY_CLEAR

```
#define LCD_DISPLAY_CLEAR 0b0000000000000001
```

Definiert in Zeile 38 der Datei [lcd_gpio.h](#).

10.11.1.6 LCD_DISPLAY_HOME

```
#define LCD_DISPLAY_HOME 0b00000000000000010
```

Definiert in Zeile 42 der Datei [lcd_gpio.h](#).

10.11.1.7 LCD_DISPLAY_OFF

```
#define LCD_DISPLAY_OFF 0b0000000000000100
```

Definiert in Zeile 35 der Datei [lcd_gpio.h](#).

10.11.1.8 LCD_DISPLAY_ON

```
#define LCD_DISPLAY_ON 0b0000000000001110
```

Definiert in Zeile 51 der Datei [lcd_gpio.h](#).

10.11.1.9 LCD_ENABLE

```
#define LCD_ENABLE _RD4 /*LCD Aktivierungssignal*/
```

Definiert in Zeile 26 der Datei [lcd_gpio.h](#).

10.11.1.10 LCD_ENTRY_MODE

```
#define LCD_ENTRY_MODE 0b00000000000000110
```

Definiert in Zeile 45 der Datei [lcd_gpio.h](#).

10.11.1.11 LCD_FUNCTION_SET

```
#define LCD_FUNCTION_SET 0b0000000000111000
```

Definiert in Zeile 32 der Datei [lcd_gpio.h](#).

10.11.1.12 LCD_R_W

```
#define LCD_R_W _RD5 /*LCD Daten Lesen oder Schreiben*/
```

Definiert in Zeile 25 der Datei [lcd_gpio.h](#).

10.11.1.13 LCD_RS

```
#define LCD_RS _RB15 /*LCD Registerauswahlsignal*/
```

Definiert in Zeile 24 der Datei [lcd_gpio.h](#).

10.11.1.14 READ_BUSY_FLAG

```
#define READ_BUSY_FLAG 0b0000000010000000
```

Definiert in Zeile 55 der Datei [lcd_gpio.h](#).

10.11.2 Dokumentation der Funktionen

10.11.2.1 lcd_clear()

```
void lcd_clear (
    void )
```

Sendet den Befehl, das LCD zurückzusetzen und den Inhalt zu löschen.

Vor dem Zugriff wird Funktion [waitForBusyLCD\(\)](#) solange blockieren, bis Busy Flag nicht gesetzt.

Definiert in Zeile [135](#) der Datei [lcd_gpio.c](#).

10.11.2.2 lcd_get_status()

```
uint8_t lcd_get_status (
    void )
```

Liest den akutellen Status des LCDs aus.

Rückgabe

8bit breiter Status, Bit 7 ist Busy Flag

PMDIN1 wird zum Puffer eingehender Daten verwendet

Definiert in Zeile [230](#) der Datei [lcd_gpio.c](#).

10.11.2.3 lcd_init()

```
void lcd_init (
    void )
```

Initialisierung des LCDs.

Drei Function Sets und anschließend die Konfiguration des LCDs. Diese beinhaltet das Funktion Set, Display off, Display clear, Entry mode und Display on.

Definiert in Zeile [22](#) der Datei [lcd_gpio.c](#).

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



10.11.2.4 lcd_set_pos()

```
void lcd_set_pos (
    int line,
    int pos )
```

Setzt die Position des Cursors.

Vor dem Zugriff wird Funktion [waitForBusyLCD\(\)](#) solange blockieren, bis Busy Flag nicht gesetzt.

Parameter

<i>line</i>	Linie, entweder 1 oder 2
<i>pos</i>	Position

Definiert in Zeile [175](#) der Datei [lcd_gpio.c](#).

10.11.2.5 lcd_write_data()

```
void lcd_write_data (
    uint8_t data )
```

Schreiben von Daten auf den Speicher des LCDs.

Vor dem Zugriff wird Funktion [waitForBusyLCD\(\)](#) solange blockieren, bis Busy Flag nicht gesetzt.

Definiert in Zeile [114](#) der Datei [lcd_gpio.c](#).

10.11.2.6 waitForBusyLCD()

```
void waitForBusyLCD (
    void )
```

Ruft [lcd_get_status\(\)](#) zyklisch auf, bis Busy Flag nicht mehr gesetzt ist.

Definiert in Zeile [251](#) der Datei [lcd_gpio.c](#).

10.11.2.7 writeStrLCD()

```
void writeStrLCD (
    const char * str )
```

Gibt einen String an der aktuellen Position im Display aus.

Parameter

str	Zeichenkette
------------	--------------

Definiert in Zeile 157 der Datei [lcd_gpio.c](#).

10.12 lcd_gpio.h

[gehe zur Dokumentation dieser Datei](#)

```

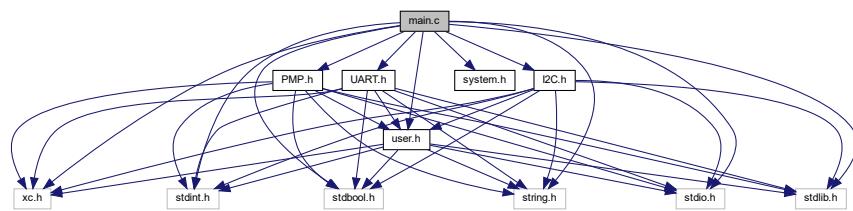
00001
00002 /* Files to Include */ 
00003
00005 #include <stdint.h>          /* Enthält uint16_t-Definition */
00006 #include <stdbool.h>           /* Enthält eine Wahr/Falsch-Definition */
00007 #include <string.h>            /* Enthält Zeichenketten */
00008 #include <stdio.h>             /* Enthält Ein - und Ausgabefunktionen */
00009 #include <stdlib.h>             /* Enthält Hilfsfunktionen */
00010
00011 #include <xc.h>                /* Jede Prozessordatei ist geschützt. */
00012
00013 #include "libpic30.h"          /* Beinhaltet Delay-Funktionen */
00014
00015
00016 /* Konstanten */ 
00017
00019 /*volatile darf nicht vom Compiler wegoptimiert werden und kann extern
00020 *verändert werden z.B. von Interruptroutine oder Timer*/
00021 #define __delay_cycles(cycles)asm volatile ("repeat %0 \n nop" : : "i" (cycles-2))
00022
00023 #define LCD_DATA(d) LATE = (LATE & 0xFF00) | d
00024 #define LCD_RS _RB15           /*LCD Registerauswahlsignal*/
00025 #define LCD_R_W _RD5            /*LCD Daten Lesen oder Schreiben*/
00026 #define LCD_ENABLE _RD4          /*LCD Aktivierungssignal*/
00027
00028 #define LCD_CMD_INIT          0b00000000000110000 /*LCD wird Initialisiert*/
00029
00030 /*Einstellen der Bewegungsrichtung von Cursor und Display.(8-Bit-Busbetrieb,
00031 *2-zeilige Anzeigemodus eingestellt) Benötigt 38us*/
00032 #define LCD_FUNCTION_SET        0b0000000000011000
00033
00034 /*Schaltet das Display aus. Benötigt 38us*/
00035 #define LCD_DISPLAY_OFF         0b0000000000001000
00036
00037 /*Löschen der Anzeigedaten. Benötigt 1.52ms*/
00038 #define LCD_DISPLAY_CLEAR       0b0000000000000001
00039
00040 /*Cursor und Anzeige werden an ihren ursprünglichen Zustand versetzt.
00041 *Benötigt 1.52ms*/
00042 #define LCD_DISPLAY_HOME        0b00000000000000010
00043
00044 /*Eingabemodus eingestellt. Cursor bewegt sich nach Rechts. Benötigt 38us*/
00045 #define LCD_ENTRY_MODE          0b0000000000000010
00046
00047 /*Schaltet Display und Cursor ein und lässt den Cursor blinken. Benötigt 38us*/
00048 /*#define LCD_DISPLAY_ON         0b0000000000001111*/
00049 /*Schaltet Display und Cursor ein und lässt den Cursor nicht mehr blinken.
00050 *Benötigt 38us*/
00051 #define LCD_DISPLAY_ON          0b0000000000001110
00052
00053 /*Definiert den IC und gibt an das dieser in Betrieb ist. Interne Operation ist
00054 *im Gange es soll gewartet werden. Benötigt 0us.*/
00055 #define READ_BUSY_FLAG          0b0000000001000000
00056
00057 /* Cursor nach rechts schieben, AC wird um 1 erhöht und gesamte Anzeige nach
00058 * rechts verschieben (der Cursor bewegt sich entsprechend der Anzeige).
00059 * In der Anzeige wird nicht gelesen oder geschrieben. Diese Anweisung wird
00060 * verwendet, um Anzeigedaten zu korrigieren oder zu suchen. Benötigt 38us. */
00061 #define CURSOR_OR_DISPLAY        0b00000000000010100
00062
00063
00064 /* Prototypen */ 
00065
00066 void lcd_init(void);
00067 void lcd_write_data(uint8_t data);
00068 void writeStrLCD(const char* str);
00069 void lcd_clear(void);
00070 void lcd_set_pos(int line, int pos);
00071 void waitForBusyLCD(void);
00072 uint8_t lcd_get_status(void);

```

10.13 main.c-Dateireferenz

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

Include-Abhängigkeitsdiagramm für main.c:



Makrodefinitionen

- #define MAIN
- #define HEARTBEAT_MS 1

Funktionen

- void print_sensor_values (void)
- double get_Temperatur (void)
- double get_Light (void)
- void measureProcessTime ()
- void display_UART_RX ()
- void send_lcd_status ()
- void display_temp_load ()
- int16_t main (void)

Variablen

- i2c_status_t status_temperatur = Error
- i2c_status_t status_licht = Error
- uint8_t write_data_buffer_temp = 0b00000000
- uint8_t write_data_buffer_light = 0b00010000
- uint8_t read_data_buffer_temp [2]
- uint8_t read_data_buffer_light [2]
- double latest_temperatur
- float latest_cpu_load
- char received_UART [20]
- int UART_RX_count

10.13.1 Makro-Dokumentation

10.13.1.1 HEARTBEAT_MS

```
#define HEARTBEAT_MS 1
```

Definiert in Zeile 32 der Datei [main.c](#).

10.13.1.2 MAIN

```
#define MAIN
```

Definiert in Zeile 31 der Datei [main.c](#).

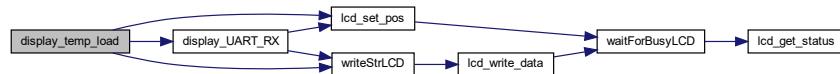
10.13.2 Dokumentation der Funktionen

10.13.2.1 display_temp_load()

```
void display_temp_load ( )
```

Definiert in Zeile 189 der Datei [main.c](#).

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



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

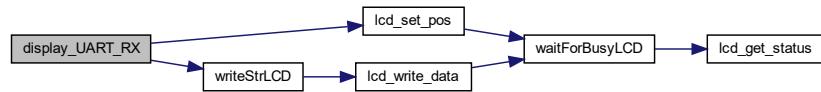


10.13.2.2 display_UART_RX()

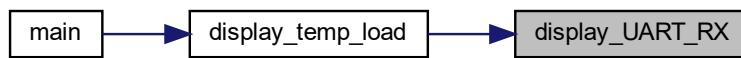
```
void display_UART_RX ( )
```

Definiert in Zeile 163 der Datei [main.c](#).

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



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



10.13.2.3 get_Light()

```
double get_Light (
    void )
```

10.13.2.4 get_Temperatur()

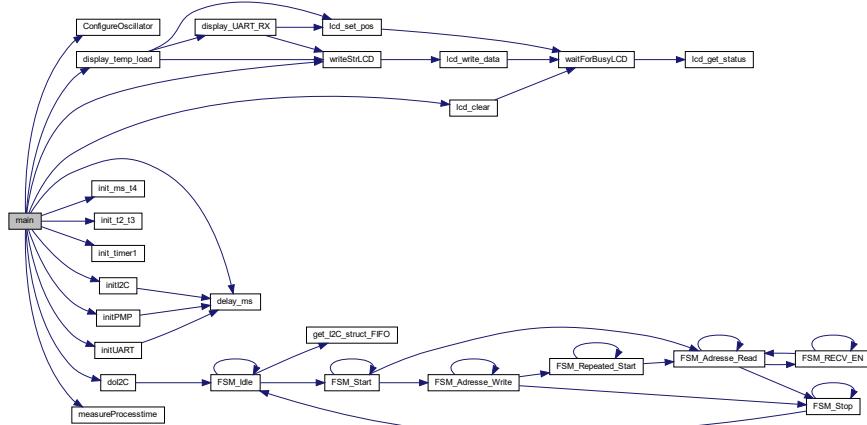
```
double get_Temperatur (
    void )
```

10.13.2.5 main()

```
int16_t main (
    void )
```

Definiert in Zeile 227 der Datei [main.c](#).

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

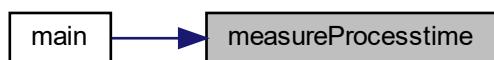


10.13.2.6 measureProcesstime()

```
void measureProcesstime ( )
```

Definiert in Zeile 137 der Datei [main.c](#).

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



10.13.2.7 print_sensor_values()

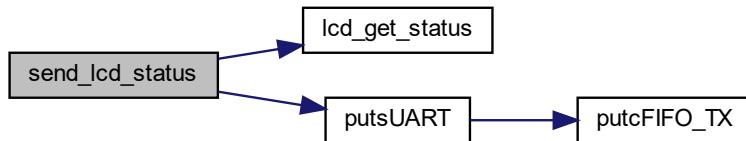
```
void print_sensor_values (
    void )
```

10.13.2.8 send_lcd_status()

```
void send_lcd_status ( )
```

Definiert in Zeile 177 der Datei [main.c](#).

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



10.13.3 Variablen-Dokumentation

10.13.3.1 latest_cpu_load

```
float latest_cpu_load
```

Definiert in Zeile 47 der Datei [main.c](#).

10.13.3.2 latest_temperatur

```
double latest_temperatur
```

Definiert in Zeile 46 der Datei [main.c](#).

10.13.3.3 read_data_buffer_light

```
uint8_t read_data_buffer_light[2]
```

Definiert in Zeile 44 der Datei [main.c](#).

10.13.3.4 `read_data_buffer_temp`

```
uint8_t read_data_buffer_temp[2]
```

Definiert in Zeile 43 der Datei [main.c](#).

10.13.3.5 `received_UART`

```
char received_UART[20]
```

Definiert in Zeile 49 der Datei [main.c](#).

10.13.3.6 `status_licht`

```
i2c_status_t status_licht = Error
```

Definiert in Zeile 39 der Datei [main.c](#).

10.13.3.7 `status_temperatur`

```
i2c_status_t status_temperatur = Error
```

Definiert in Zeile 38 der Datei [main.c](#).

10.13.3.8 `UART_RX_count`

```
int UART_RX_count
```

Definiert in Zeile 50 der Datei [main.c](#).

10.13.3.9 `write_data_buffer_light`

```
uint8_t write_data_buffer_light = 0b00010000
```

Definiert in Zeile 42 der Datei [main.c](#).

10.13.3.10 write_data_buffer_temp

```
uint8_t write_data_buffer_temp = 0b00000000
```

Definiert in Zeile 41 der Datei [main.c](#).

10.14 main.c

[gehe zur Dokumentation dieser Datei](#)

```
00001 /*TODO
00002     Doku mit DoxyGen
00003 *
00004 */
00005
00006
00007 /* Files to Include *//
00008
00009 #include <xc.h>          /* Jede Prozessordatei ist geschützt. */
00010
00011 #include <stdint.h>        /* Enthält uint16_t-Definition */
00012 #include <stdbool.h>        /* Enthält eine Wahr/Falsch-Definition */
00013 #include <string.h>         /* Enthält Zeichenketten */
00014 #include <stdio.h>          /* Enthält Ein - und Ausgabefunktionen */
00015 #include <stdlib.h>         /* Enthält Hilfsfunktionen */
00016
00017 #include "system.h"         /* System - Funktion/Parameter */
00018 #include "user.h"           /* Benutzer - Funktion/Parameter */
00019
00020
00021 /* Beinhaltet Konstanten, Typdefs, globale Variablen und Prototypen für main */
00022 #include "UART.h"
00023 #include "I2C.h"
00024 //##include "lcd_gpio.h"
00025 #include "PMP.h"
00026
00027
00028 /* Konstanten *//
00029
00030 #define MAIN
00031 #define HEARTBEAT_MS 1
00032
00033
00034
00035 /* Global Variable Declaration *//
00036
00037 i2c_status_t status_temperatur = Error;
00038 i2c_status_t status_licht = Error;
00039
00040 uint8_t write_data_buffer_temp = 0b00000000;
00041 uint8_t write_data_buffer_light = 0b00010000;
00042 uint8_t read_data_buffer_temp[2];
00043 uint8_t read_data_buffer_light[2];
00044
00045 double latest_temperatur;
00046 float latest_cpu_load;
00047
00048
00049 char received_UART[20];
00050 int UART_RX_count;
00051
00052
00053 /* Prototypen *//
00054
00055 void print_sensor_values(void);
00056
00057 double get_Temperatur(void);
00058 double get_Light(void);
00059
00060
00061
00062 /* Funktionen *//
00063
00064 #if 0
00065 void print_sensor_values()
00066 {
00067     static int count = 0;
00068
00069     //Temperatur
00070     if (status_temperatur == Finished)
00071     {
00072         //Ausgabe per UART
00073         char str[32];
00074
00075         str[0] = 'T';
00076         str[1] = 'e';
00077         str[2] = 'm';
00078         str[3] = 'e';
00079         str[4] = 'r';
00080         str[5] = 'a';
00081         str[6] = 't';
00082         str[7] = 'u';
00083         str[8] = 'r';
00084         str[9] = 'u';
00085         str[10] = 'r';
00086         str[11] = 'u';
00087         str[12] = 'r';
00088         str[13] = 'u';
00089         str[14] = 'r';
00090         str[15] = 'u';
00091         str[16] = 'r';
00092         str[17] = 'u';
00093         str[18] = 'r';
00094         str[19] = 'u';
00095         str[20] = 'r';
00096         str[21] = 'u';
00097         str[22] = 'r';
00098         str[23] = 'u';
00099         str[24] = 'r';
00100         str[25] = 'u';
00101         str[26] = 'r';
00102         str[27] = 'u';
00103         str[28] = 'r';
00104         str[29] = 'u';
00105         str[30] = 'r';
00106         str[31] = '\0';
00107
00108     }
00109 }
```

```

00078     sprintf(str,"Temperatur: %.1f Grad",get_Temperatur());
00079     putsUART(str);
00080     char lf[2];
00081     sprintf(lf, "\n");
00082     putsUART(lf);
00083
00084     latest_temperatur=get_Temperatur();
00085     status_temperatur=Pending;
00086 }
00087
00088 //Licht
00089 if (status_licht == Finished)
00090 {
00091     char str[16];
00092     sprintf(str,"Licht: %.1f lux",get_Light());
00093     putsUART(str);
00094     char lf[2];
00095     sprintf(lf, "\n");
00096     putsUART(lf);
00097     status_licht=Pending;
00098 }
00099
00100 if (status_licht == Error || status_temperatur == Error)
00101 {
00102     if (count >= 1000)
00103     {
00104         count = 0;
00105         char str[32];
00106         sprintf(str, "Fehler beim Auslesen!");
00107         putsUART(str);
00108         char lf[2];
00109         sprintf(lf, "\n");
00110         putsUART(lf);
00111     }
00112
00113     else
00114     {
00115         count++;
00116     }
00117 }
00118
00119 } /* print_sensor_values() */
00120 double get_Temperatur(void)
00121 {
00122     double temp = read_data_buffer_temp[0] << 8 | read_data_buffer_temp[1];
00123     temp = temp / 256;
00124     return temp;
00125
00126 }/*get_Temperatur()*/
00127 double get_Light(void)
00128 {
00129     double light = read_data_buffer_light[0] << 8 | read_data_buffer_light[1];
00130     light = light / 1.2;
00131     return light;
00132
00133 }/*get_Light()*/
00134 #endif
00135
00136
00137 void measureProcesstime()
00138 {
00139     static uint16_t time = 0;
00140     time++;
00141
00142     if (time >= 10000) //Alle 10s Auslastung berechnen
00143     {
00144         float Auslastung;
00145         uint16_t lsw = TMR2;
00146         uint16_t msw = TMR3HLD;
00147
00148         float Auslastung1 = (float)lsw * 10;
00149         float Auslastung2 = (float)msw * 65535 * 10;
00150         Auslastung = (Auslastung1 + Auslastung2) / (FCY);
00151
00152         latest_cpu_load=Auslastung;
00153
00154         time = 0;
00155         TMR3HLD = 0;
00156         TMR3 = 0; // Clear 32-bit Timer (msw)
00157         TMR2 = 0;
00158     }
00159
00160 }/*measureProcesstime()*/
00161
00162
00163 void display_UART_RX()
00164 {

```

```

00165     if (UART_RX_count > 0)
00166     {
00167         lcd_set_pos(2,1);
00168         writeStrLCD("                ");
00169         lcd_set_pos(2,1);
00170         writeStrLCD(received_UART);
00171         memset(received_UART,0,sizeof(received_UART));
00172         UART_RX_count = 0;
00173     }
00174
00175 /*display_UART_RX()*/
00176
00177 void send_lcd_status()
00178 {
00179     char str[32];
00180     sprintf(str,"Status: %02X",lcd_get_status());
00181     putsUART(str);
00182     char lf[2];
00183     sprintf(lf, "\n");
00184     putsUART(lf);
00185
00186 /*send_lcd_status()*/
00187
00188
00189 void display_temp_load()
00190 {
00191     static uint16_t time = 0;
00192     static bool switch_value = 1;
00193     time++;
00194
00195     if (time >= 2000) //Alle 2s Refresh
00196     {
00197         time = 0;
00198
00199         if (switch_value) //Temperatur
00200         {
00201             char str[32];
00202             sprintf(str,"Temperatur: %.1f",latest_temperatur);
00203             lcd_set_pos(1,1);
00204             writeStrLCD(str);
00205             switch_value = 0;
00206         }
00207
00208         else //Auslastung
00209         {
00210             char str[32];
00211             sprintf(str,"Auslastung: %.2f", (double)latest_cpu_load);
00212             lcd_set_pos(1,1);
00213             writeStrLCD(str);
00214             switch_value = 1;
00215         }
00216
00217         display_UART_RX();
00218     }
00219
00220 /*display_temp_load()*/
00221
00222
00223
00224 /* Main Program */ *
00225
00226 int16_t main(void)
00227 {
00228     uint16_t Count = 0;
00229
00230     ConfigureOscillator();
00231     initUART();
00232     init_timer1();           //Jede Sekunde I2C Anfrage
00233     init_ms_t4();           //Heartbeat in Superloop
00234     init_t2_t3();
00235     initI2C();
00236     //lcd_init();
00237     initPMP();
00238
00239     writeStrLCD("Hello World");
00240     delay_ms(2000);
00241     lcd_clear();
00242
00243     _RPOUT_U1TX;           //UART Pin Mapping
00244     RPINR18bits.U1RXR = 0b1011000;
00245
00246     _TRISF0 = 0;            //Für Auslastungsberechnung
00247     _T2CKR = 96;            //LATF0 auf Gated Timer Input mappen
00248
00249     _TRISE8 = 1;            //Uart RX als Input
00250     _ANSE8 = 0;
00251
00252

```

```

00253     while(1)
00254     {
00255         _LATF0 = 0;           //Gated Timer stoppen
00256
00257         if(_T4IF)
00258         {
00259             _T4IF = 0;
00260             Count++;
00261
00262             if (Count >= HEARTBEAT_MS)
00263             {
00264                 _LATF0 = 1;       //Gated Timer starten
00265                 Count = 0;
00266
00267                 doI2C();
00268                 measureProcesstime();
00269                 display_temp_load();
00270             }
00271         }
00272     }
00273 }/*main()*/

```

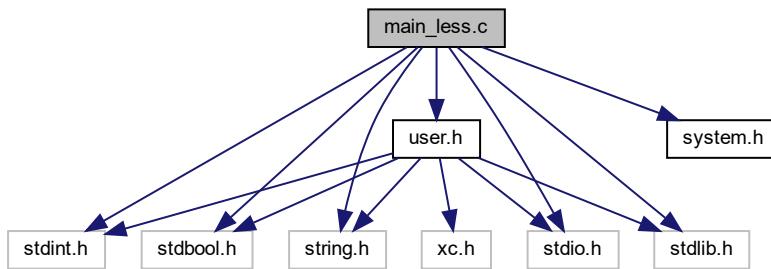
10.15 main_less.c-Dateireferenz

```

#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}

10.15.1 Makro-Dokumentation

10.15.1.1 BAUDRATE

```
#define BAUDRATE 9600
```

Definiert in Zeile 32 der Datei [main_less.c](#).

10.15.1.2 BRGVAL

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

Definiert in Zeile 33 der Datei [main_less.c](#).

10.15.1.3 BUFFER_FAIL

```
#define BUFFER_FAIL 0
```

Definiert in Zeile 36 der Datei [main_less.c](#).

10.15.1.4 BUFFER_SIZE

```
#define BUFFER_SIZE 128
```

Definiert in Zeile 38 der Datei [main_less.c](#).

10.15.1.5 BUFFER_SUCCESS

```
#define BUFFER_SUCCESS 1
```

Definiert in Zeile 37 der Datei [main_less.c](#).

10.15.1.6 HEARTBEAT_MS

```
#define HEARTBEAT_MS 1
```

Definiert in Zeile 29 der Datei [main_less.c](#).

10.15.1.7 I2C_SCL

```
#define I2C_SCL _RA2
```

Definiert in Zeile 42 der Datei [main_less.c](#).

10.15.1.8 I2C_SCL_TRIS

```
#define I2C_SCL_TRIS _TRISA2
```

Definiert in Zeile 44 der Datei [main_less.c](#).

10.15.1.9 I2C_SDA

```
#define I2C_SDA _RA3
```

Definiert in Zeile 43 der Datei [main_less.c](#).

10.15.1.10 I2C_SDA_TRIS

```
#define I2C_SDA_TRIS _TRISA3
```

Definiert in Zeile 45 der Datei [main_less.c](#).

10.15.2 Dokumentation der benutzerdefinierten Typen

10.15.2.1 StateFunc

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

Definiert in Zeile 58 der Datei [main_less.c](#).

10.15.3 Dokumentation der Funktionen

10.15.3.1 _T1Interrupt()

```
void _T1Interrupt (
    void )
```

Definiert in Zeile 91 der Datei [main_less.c](#).

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



10.15.3.2 _U1TXInterrupt()

```
void _U1TXInterrupt (
    void )
```

Definiert in Zeile 136 der Datei [main_less.c](#).

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



10.15.3.3 delay_ms()

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

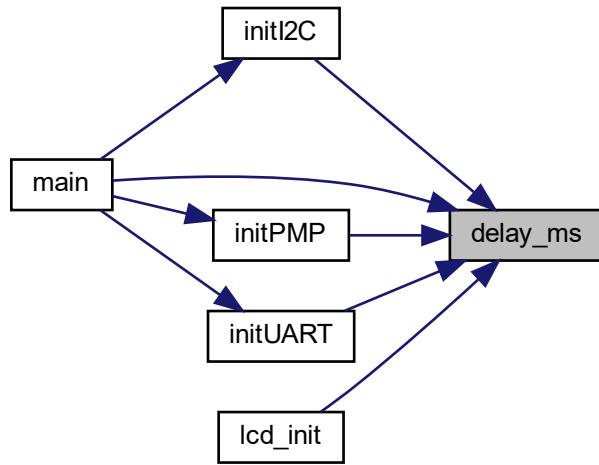
Delay in ms Blockierende Delay Funktion, eventuell nicht perfekte Verzögerung mit Hilfe einer for Schleife.

Parameter

in	<i>milliseconds</i>	Verzögerungszeit in millisekunden
----	---------------------	-----------------------------------

Definiert in Zeile 85 der Datei [main_less.c](#).

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

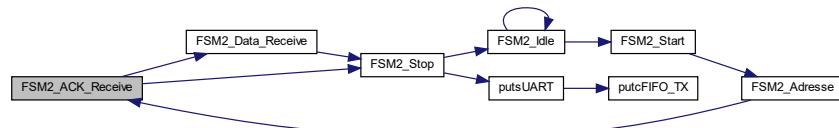


10.15.3.4 FSM2_ACK_Receive()

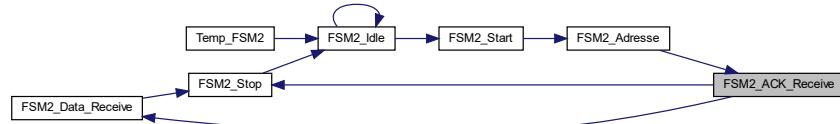
```
void * FSM2_ACK_Receive (
    void )
```

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:

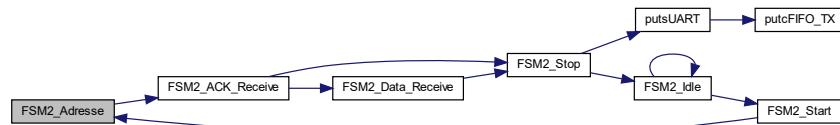


10.15.3.5 FSM2_Adresse()

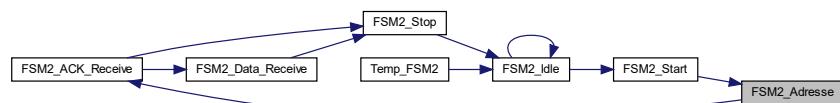
```
void * FSM2_Adresse (
    void )
```

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:

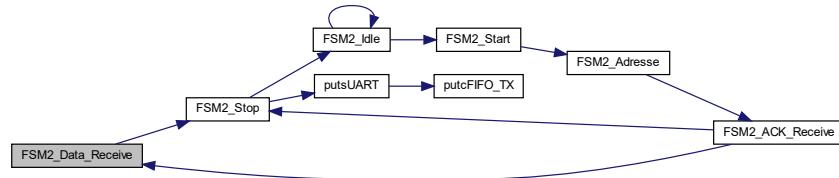


10.15.3.6 FSM2_Data_Receive()

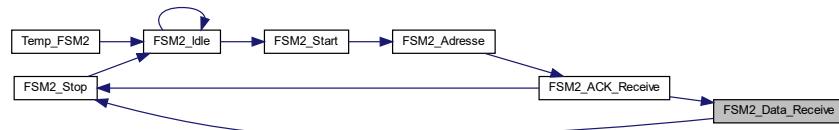
```
void * FSM2_Data_Receive (
    void )
```

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:

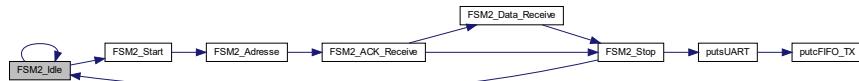


10.15.3.7 FSM2_Idle()

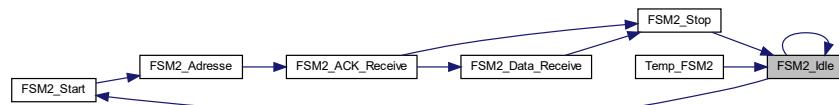
```
void * FSM2_Idle (
    void )
```

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:

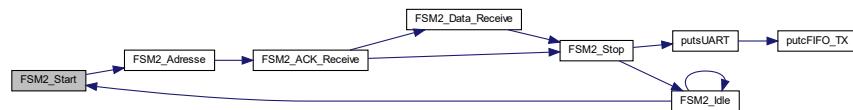


10.15.3.8 FSM2_Start()

```
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:

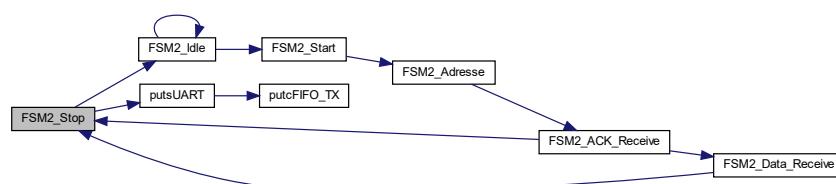


10.15.3.9 FSM2_Stop()

```
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:



10.15.3.10 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:



10.15.3.11 init_ms_t4()

```
void init_ms_t4 (
    void )
```

Definiert in Zeile 123 der Datei [system.c](#).

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

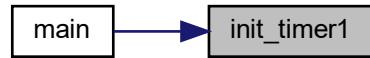


10.15.3.12 init_timer1()

```
void init_timer1 (
    void )
```

Definiert in Zeile 210 der Datei [main_less.c](#).

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



10.15.3.13 initI2C()

```
void initI2C (
    void )
```

Initialisiert die I2C-Kommunikation.

Einschalten I2C mit eigenem Workaround, I2C Peripheriemodul kann hier leider nicht verwendet werden. Pins wie einen Open-Kollektor-Treiber verwenden, d.h. 1 - hochohmig, 0 wird getrieben

Aktiviert das I2C Modul und konfiguriert die Pins SDAx und SCLx als serielle PORT-Pins.

Definiert in Zeile [234](#) der Datei [main_less.c](#).

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



10.15.3.14 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:

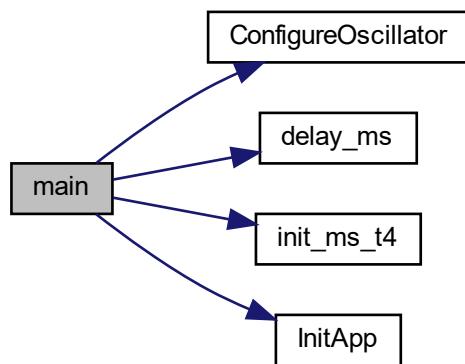


10.15.3.15 main()

```
int16_t main (
    void )
```

Definiert in Zeile 376 der Datei [main_less.c](#).

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

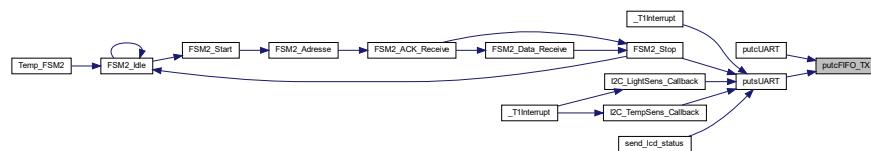


10.15.3.16 putcFIFO_TX()

```
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:



10.15.3.17 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:



10.15.3.18 putsUART()

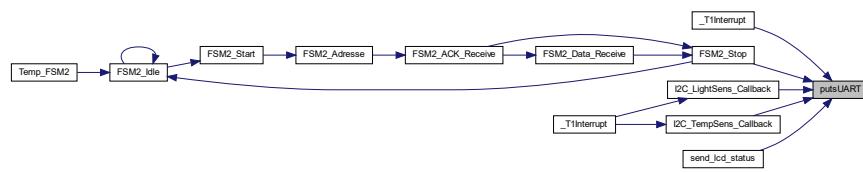
```
int16_t putsUART (
    const char * str )
```

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:

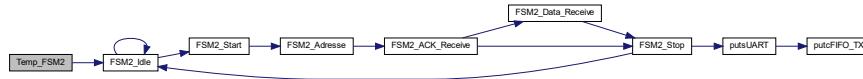


10.15.3.19 Temp_FSM2()

```
void Temp_FSM2 ( void )
```

Definiert in Zeile 227 der Datei main_less.c.

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



10.15.4 Variablen-Dokumentation

10.15.4.1 data

```
uint8_t data[2]
```

Definiert in Zeile 46 der Datei [main_less.c](#).

10.15.4.2 DELAY_ANPASSUNG

```
uint32_t DELAY_ANPASSUNG
```

Definiert in Zeile 39 der Datei [main_less.c](#).

10.15.4.3 FIFO

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

Definiert in Zeile 56 der Datei [main_less.c](#).

10.16 main_less.c

[gehe zur Dokumentation dieser Datei](#)

```
00001
00002 /* Files to Include */                                                 */
00003
00004 /* Device header file */
00005 #if defined(__XC16__)
00006     #include <xc.h>
00007 #elif defined(__C30__)
00008     #if defined(__dsPIC33E__)
00009         #include <p33Exxxx.h>
00010     #elif defined(__dsPIC33F__)
00011         #include <p33Fxxxx.h>
00012     #endif
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_RX(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*****
00080 void delay_ms(uint16_t milliseconds) {
00081     uint32_t i=0;
00082     for (i=0;i<(DELAY_ANPASSUNG*(uint32_t)milliseconds);i++) {
00083         }
00084     }
00085
00086
00087
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.UTXBKR = 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     _UITXIE = 1; // Enable UART TX interrupt
00133
00134 }
00135
00136 void _attribute_((__interrupt__) _U1TXInterrupt(void)
00137 {
00138     _UITXIF = 0; // Clear TX Interrupt flag
00139
00140     getcFIFO_RX(&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_RX(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; // Interrupt 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
00239     // werden
00240     I2C_SDA_TRIS = 1; // Pins wie einen Open-Kollektor-Treiber verwenden, d.h. 1 - hochohmig, 0
00241     // wird getrieben
00242     I2C_SCL_TRIS = 1;
00243     I2C_SDA = 0;
00244     I2C_SCL = 0;
00245
00246     int j;
00247     for (j=0; j<=9; j++) // takten bis min 1 Byte
00248     {
00249         I2C_SCL_TRIS = 0; delay_ms(1); // 5 us wären ausreichend ...100 kBaud
00250         I2C_SCL_TRIS = 1; delay_ms(1);
00251     }
00252     // Start Condition senden
00253     I2C_SCL_TRIS = 0; delay_ms(1);
00254     I2C_SDA_TRIS = 0; delay_ms(1);
00255     // Stop Condition senden
00256     I2C_SCL_TRIS = 1; delay_ms(1);
00257     I2C_SDA_TRIS = 1; delay_ms(1);
00258
00259     // Nun I2C erst anschalten
00260     _MI2C2IF = 0; //Interrupt falls noetig
00261     _MI2C2IE = 0;
00262     I2C2CONbits.I2CEN = 1;
00263
00264     //Sensor Pointer auf TEMP Register setzen
00265     I2C2CONbits.SEN=1; //start
00266     while(I2C2CONbits.SEN==1){}
00267
00268     //Tx Device address + Write bit
00269     I2C2TRN=0b10010000;
00270     while(I2C2STATbits.TRSTAT==1){}
00271
00272     if (I2C2STATbits.ACKSTAT==1){ //if NACK received, generate stop condition and exit
00273         I2C2STATbits.ACKSTAT=0;
00274         I2C2CONbits.PEN=1;
00275         while(I2C2CONbits.PEN==1){} //wait for the stop interrupt;
00276         return;
00277     }
00278
00279     //Tx Register Address
00280     I2C2TRN=0b00000000; //Pointer auf TEMP REGISTER setzen
00281     while(I2C2STATbits.TRSTAT==1){}
00282
00283     if (I2C2STATbits.ACKSTAT==1){ //if NACK received, generate stop condition and exit
00284         I2C2STATbits.ACKSTAT=0;
00285         I2C2CONbits.PEN=1;
00286         while(I2C2CONbits.PEN==1){} //wait for the stop interrupt;
00287         return;
00288     }
00289     I2C2CONbits.PEN=1; //stop
00290     while(I2C2CONbits.PEN==1){} //wait for the stop interrupt
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
00372
00373
00374
00375 int16_t main(void)
00376 {
00377 DELAY_ANPASSUNG = ((SYS_FREQ/96)*2180ull)/1000000ull; //Berechnung der Delay Anpassung
00378 //uint16_t Count = 0;
00379 /* Configure the oscillator for the device */
00380 ConfigureOscillator();
00381 /* Initialize IO ports and peripherals */
00382 InitApp();
00383
00384 //initUART();
00385 //init_timer1();
00386 init_ms_t4();
00387 //initI2C();
00388
00389
00390 TRISBbits.TRISB8 = 0; //LED0 als Ausgang
00391 ANSELBbits.ANSB8 = 0; //LED0 als Digitaler Ausgang
00392
00393 TRISBbits.TRISB9 = 0; //LED als Ausgang
00394 ANSELBbits.ANSB9 = 0;
00395
00396 //Taster als Eingänge
00397 _TRISG12 = 1;
00398 //Pull-up Widerstände einschalten
00399 _CNPUG12 = 1;
00400
00401
00402 _RPO66R = _RPOUT_U1TX; //UART Pin Mapping
00403 RPINR18bits.U1RXR = 0b1011000;
00404 /* TODO <INSERT USER APPLICATION CODE HERE> */
00405
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 }

```

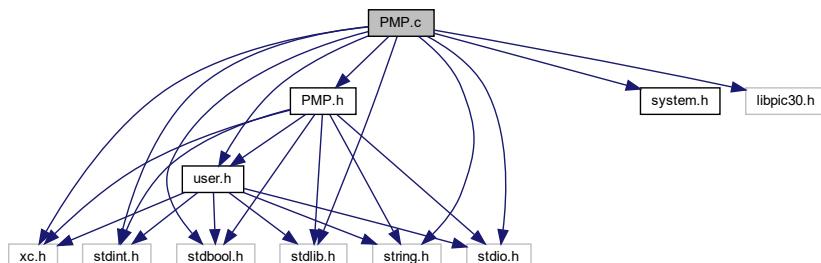
10.17 PMP.c-Dateireferenz

```

#include <xc.h>
#include <stdint.h>
#include <stdbool.h>
#include <string.h>
#include <stdio.h>
#include <stdlib.h>
#include "system.h"
#include "user.h"
#include "libpic30.h"
#include "PMP.h"

```

Include-Abhängigkeitsdiagramm für PMP.c:



Funktionen

- void **initPMP** (void)
Initialisierung des PMPs.
- uint8_t **lcd_get_status** (void)
Liest den aktuellen Status des LCDs aus.
- void **waitForBusyLCD** (void)
Ruft lcd_get_status() zyklisch auf, bis Busy Flag nicht mehr gesetzt ist.
- void **lcd_write_data** (uint8_t data)
Schreiben von Daten auf den Speicher des LCDs.
- void **writeStrLCD** (const char *str)
Gibt einen String an der aktuellen Position im Display aus.
- void **lcd_clear** (void)
Sendet den Befehl, das LCD zurückzusetzen und den Inhalt zu löschen.
- void **lcd_set_pos** (int line, int pos)
Setzt die Position des Cursors.

10.17.1 Dokumentation der Funktionen

10.17.1.1 initPMP()

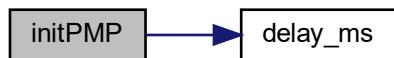
```
void initPMP (
    void )
```

Initialisierung des PMPs.

Drei Function Sets und anschließend die Konfiguration des LCDs. Diese beinhaltet das Funktion Set, Display off, Display clear, Entry mode und Display on.

Definiert in Zeile 29 der Datei [PMP.c](#).

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



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



10.17.1.2 lcd_clear()

```
void lcd_clear (
    void )
```

Sendet den Befehl, das LCD zurückzusetzen und den Inhalt zu löschen.

Vor dem Zugriff wird Funktion [waitForBusyLCD\(\)](#) solange blockieren, bis Busy Flag nicht gesetzt.

Definiert in Zeile 198 der Datei [PMP.c](#).

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



10.17.1.3 lcd_get_status()

```
uint8_t lcd_get_status (
    void )
```

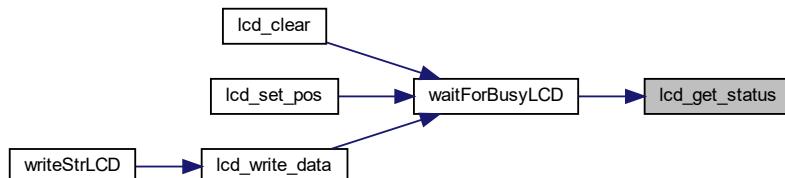
Liest den aktuellen Status des LCDs aus.

Rückgabe

PMDIN1 wird zum Puffer eingehender Daten verwendet

Definiert in Zeile 126 der Datei [PMP.c](#).

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



10.17.1.4 lcd_set_pos()

```
void lcd_set_pos (
    int line,
    int pos )
```

Setzt die Position des Cursors.

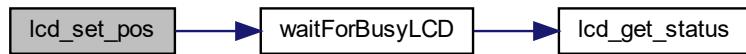
Vor dem Zugriff wird Funktion [waitForBusyLCD\(\)](#) solange blockieren, bis Busy Flag nicht gesetzt.

Parameter

<i>line</i>	Linie, entweder 1 oder 2
<i>pos</i>	Position

Definiert in Zeile [216](#) der Datei [PMP.c](#).

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



10.17.1.5 lcd_write_data()

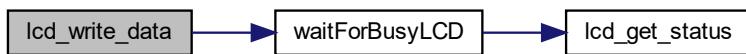
```
void lcd_write_data (
    uint8_t data )
```

Schreiben von Daten auf den Speicher des LCDs.

Vor dem Zugriff wird Funktion [waitForBusyLCD\(\)](#) solange blockieren, bis Busy Flag nicht gesetzt.

Definiert in Zeile [166](#) der Datei [PMP.c](#).

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



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



10.17.1.6 waitForBusyLCD()

```
void waitForBusyLCD (
    void )
```

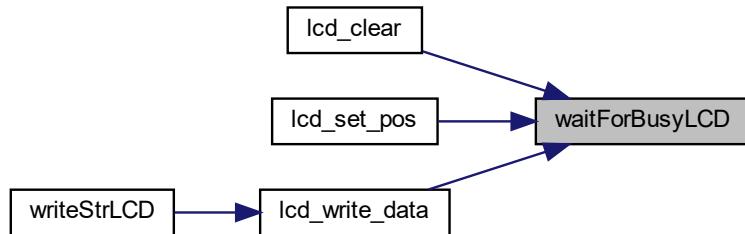
Ruft [lcd_get_status\(\)](#) zyklisch auf, bis Busy Flag nicht mehr gesetzt ist.

Definiert in Zeile [149](#) der Datei [PMP.c](#).

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



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



10.17.1.7 writeStrLCD()

```
void writeStrLCD (
    const char * str )
```

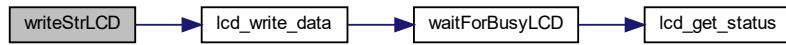
Gibt einen String an der aktuellen Position im Display aus.

Parameter

<code>str</code>	Zeichenkette
------------------	--------------

Definiert in Zeile [180](#) der Datei [PMP.c](#).

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



10.18 PMP.c

[gehe zur Dokumentation dieser Datei](#)

```

00001
00002 /* Files to Include
00003
00005 #include <xc.h>           /* Jede Prozessordatei ist geschützt.
00006
00007 #include <stdint.h>         /* Enthält uint16_t-Definition
00008 #include <stdbool.h>         /* Enthält eine Wahr/Falsch-Definition
00009 #include <string.h>          /* Enthält Zeichenketten
00010 #include <stdio.h>           /* Enthält Ein- und Ausgabefunktionen
00011 #include <stdlib.h>          /* Enthält Hilfsfunktionen
00012
00013 #include "system.h"          /* System - Funktion/Parameter
00014 #include "user.h"            /* Benutzer - Funktion/Parameter
00015
00016 #include "libpic30.h"         /* Beinhaltet Delay-Funktionen
00017 #include "PMP.h"             /* Behinhaltet Konstanten und Prototypen
00018
00019
00020
00021 /* Funktionen
00022
00029 void initPMP(void)
00030 {
00031     //Alle Ausgänge als Digital
00032     _ANSB15 = 0;
00033     ANSELE &= 0xFF00;
00034
00035     //CON
00036     PMCONbits.PMPEN = 1;
00037     PMCONbits.PSIDL = 0;
00038     PMCONbits.ADRMUX = 0b00;
00039     PMCONbits.PTBEN = 0;
00040     PMCONbits.PTWREN = 1;
00041     PMCONbits.PTRDEN = 1;
00042     PMCONbits.CSF = 0b00;
00043     PMCONbits.ALPI = 1;
00044     PMCONbits.CS2P = 1;
00045     PMCONbits.CS1P = 1;
00046     PMCONbits.BEP = 1;
00047     PMCONbits.WRSP = 1;
00048     PMCONbits.RDSP = 1;
00049
00050     //MODE
00051     PMMODEbits.IRQM = 0b00;
00052     PMMODEbits.INCM = 0b00;
00053     PMMODEbits.MODE16 = 0;
00054     PMMODEbits.MODE = 0b11;    //Master Mode 1
00055     PMMODEbits.WAITB = 0b11;
00056     PMMODEbits.WAITM = 0b1111;
00057     PMMODEbits.WAITE = 0b11;
00058
00059     //ADDRESS ENABLE
00060
00062 /* Parallel Master Port Address Enable Register (PMAEN). Dieses Register
00063 * steuert den Betrieb der Adress- und Chip-Select-Pins, die mit dem PMP
00064 * Modul zugeordnet sind.*/
00065 PMAEN = 0x0001;           // PMA0 enabled
00066
00067 /* Pad Configuration Control Register (PADCFIG1). Das PADCFIG1 steuert,
00068 * welcher digitale Eingangspuffer vom PMP-Modul verwendet wird.
00069 * Das PMPTL Bit ermöglicht dem Benutzer die Auswahl zwischen
00070 * Transistor-Transistor-Logik (TTL) und Schmitt-Trigger
00071 * (ST)-Eingangspuffern für eine bessere Kompatibilität mit externen
  
```

```

00072     * Schaltungen. PMPTTL = 0 ist die Standardkonfiguration und wählt die
00073     * ST-Puffer aus.*/
00074 PMDCFG1bits.PMPTTL = 0;
00075 delay_ms(40);           //40 ms warten
00076
00077
00079 /* Parallel Master Port Address Register (PMADDR) Dieses Register enthält
00080 * die Adresse, an die die ausgehenden Daten geschrieben werden sollen,
00081 * sowie die Chip Select Steuerbits für die Adressierung von parallelen
00082 * Slave-Geräten.*/
00083 PMADDR = 0;             //RS auf 0
00084
00085 /* Der Register Parallel Master Port Data Input 1 (PMDIN1) dient der
00086 * Pufferung eingehender Daten.*/
00087 PMDIN1 = LCD_CMD_INIT;    //Function Set 1
00088 __delay_us(4100);        //4.1 ms warten
00089
00090 PMDIN1 = LCD_CMD_INIT;    //Function Set 2
00091 __delay_us(100);         //100 us warten
00092
00093 PMDIN1 = LCD_CMD_INIT;    //Function Set 3
00094 __delay_us(38);          //LCD_CMD_INIT benötigt 38 us zum Ausführen
00095
00096
00098 /*LCD_FUNCTION_SET stellt die Bewegungsrichtig von Cursor und Display ein.*/
00099 PMDIN1 = LCD_FUNCTION_SET;
00100 __delay_us(38);          //LCD_FUNCTION_SET benötigt 38 us zum Ausführen
00101
00102 /*LCD_DISPLAY_OFF schaltet das Display aus.*/
00103 PMDIN1 = LCD_DISPLAY_OFF;
00104 __delay_us(38);          //LCD_DISPLAY_OFF benötigt 38 us zum Ausführen
00105
00106 /*LCD_DISPLAY_CLEAR löscht die Anzeigedaten. LCD_DISPLAY_CLEAR benötigt
00107 * 1520 us zum Ausführen*/
00108 PMDIN1 = LCD_DISPLAY_CLEAR;
00109 __delay_us(1520);
00110
00111 /*LCD_ENTRY_MODE stellt den Eingabemodus ein. Der Cursor bewegt sich dabei
00112 *nach rechts.*/
00113 PMDIN1 = LCD_ENTRY_MODE;
00114 __delay_us(38);          //LCD_ENTRY_MODE benötigt 38 us zum Ausführen
00115
00116 /*LCD_DISPLAY_ON schaltet das Display ein.*/
00117 PMDIN1 = LCD_DISPLAY_ON;
00118 __delay_us(38);          //LCD_DISPLAY_ON benötigt 38 us zum Ausführen
00119
00120 }/*initPMP()*/
00121
00126 uint8_t lcd_get_status(void)
00127 {
00128     uint8_t dummy;
00129
00130     /* Parallel Master Port Mode Register (PMMODE). Dieses Register wählt den
00131     * Master/Slave-Modus aus und konfiguriert die Betriebsmodi des PMP-Moduls.
00132     * Dieses Register enthält das universelle Status-Flag BUSY, das in
00133     * Master-Modi verwendet wird, um anzugeben, dass eine Operation durch das
00134     * Modul im Gange ist*/
00135     while( PMMODEbits.BUSY);    //Warten bis PMP bereit
00136     PMADDR = 0;
00137
00138     dummy = PMDIN1;            //lesen anstoßen durch dummy read
00139
00140     while( PMMODEbits.BUSY);
00141     return (PMDIN1);
00142
00143 }/*lcd_get_status()*/
00144
00145
00149 void waitForBusyLCD(void)
00150 {
00151     /* READ_BUSY_FLAG definiert den IC und gibt an das dieser in Betrieb ist.
00152     * Interne Operation ist im Gange es soll gewartet werden.*/
00153     while((lcd_get_status() & READ_BUSY_FLAG))
00154     {
00155
00156     }
00157     return;
00158
00159 }/*waitForBusyLCD()*/
00160
00161
00166 void lcd_write_data(uint8_t data)
00167 {
00168     waitForBusyLCD();
00169     while( PMMODEbits.BUSY );
00170     PMADDR = 1;
00171     PMDIN1 = data;

```

```

00172 }/*lcd_write_data()*/
00174
00175
00176 void writeStrLCD(const char* str)
00177 {
00178     uint8_t i = 0;
00179
00180     while (str[i]!=0)
00181     {
00182         lcd_write_data((str[i]));
00183         i++;
00184     }
00185
00186 }/*writeStrLCD()*/
00187
00188
00189 void lcd_clear(void)
00190 {
00191     waitForBusyLCD();
00192     while( PMODEbits.BUSY );
00193     PMADDR = 0;
00194
00195     /*LCD_DISPLAY_CLEAR löscht die Anzeigedaten.*/
00196     PMDIN1 = LCD_DISPLAY_CLEAR;
00197
00198 }/*lcd_clear()*/
00199
00200
00201
00202
00203
00204
00205
00206
00207
00208
00209
00210 void lcd_set_pos(int line, int pos)
00211 {
00212     /*Display auf Home Position*/
00213     waitForBusyLCD();
00214     while( PMODEbits.BUSY );
00215     PMADDR = 0;
00216
00217     /* LCD_DISPLAY_HOME setzt Cursor und Anzeige an ihren ursprünglichen Zustand
00218      * zurück.*/
00219     PMDIN1 = LCD_DISPLAY_HOME;
00220
00221     /*Berechnung wieviel geshiftet werden muss*/
00222     int i = 0;
00223     int to_shift = 0;
00224
00225     if (line == 1)           //Erste Linie
00226     {
00227         to_shift = pos;
00228     }
00229
00230     else                   //Zweite Linie
00231     {
00232         to_shift = pos + 40;
00233     }
00234
00235     for(i = 0; i < to_shift - 1; i++)
00236     {
00237         waitForBusyLCD();
00238         while( PMODEbits.BUSY );
00239         PMADDR = 0;
00240
00241         /*CURSOR_OR_DISPLAY schiebt Cursor und Anzeige nach rechts (der Cursor
00242          *bewegt sich entsprechend der Anzeige).*/
00243         PMDIN1 = CURSOR_OR_DISPLAY;
00244     }
00245
00246
00247 }/*lcd_set_pos()*/
00248
00249
00250
00251
00252 }/*lcd_set_pos()*/

```

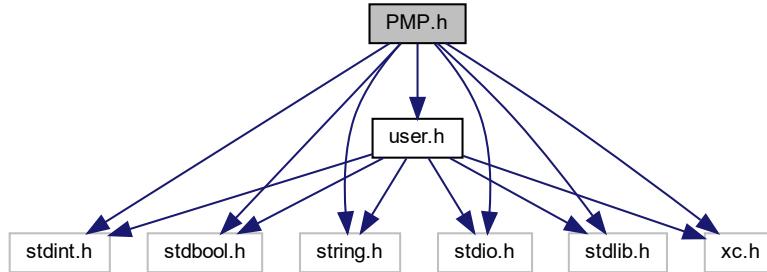
10.19 PMP.h-Dateireferenz

```

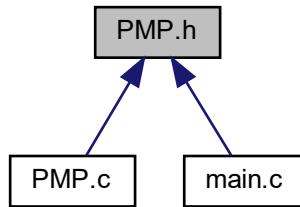
#include "user.h"
#include <stdint.h>
#include <stdbool.h>
#include <string.h>
#include <stdio.h>
#include <stdlib.h>
#include <xc.h>

```

Include-Abhängigkeitsdiagramm für PMP.h:



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



Makrodefinitionen

- `#define __delay_cycles(cycles) asm volatile ("repeat #%`*0* \n `nop" : : "i" (cycles-2))`
- `#define LCD_DATA(d) LATE = (LATE & 0xFF00) | d`
- `#define LCD_RS_RB15 /*LCD Registerauswahlsignal*/`
- `#define LCD_R_W_RD5 /*LCD Daten Lesen oder Schreiben*/`
- `#define LCD_ENABLE_RD4 /*LCD Aktivierungssignal*/`
- `#define LCD_CMD_INIT 0b00000000000110000 /*LCD wird Initialisiert*/`
- `#define LCD_FUNCTION_SET 0b0000000000011100`
- `#define LCD_DISPLAY_OFF 0b0000000000001000`
- `#define LCD_DISPLAY_CLEAR 0b000000000000000001`
- `#define LCD_DISPLAY_HOME 0b0000000000000000010`
- `#define LCD_ENTRY_MODE 0b00000000000000110`
- `#define LCD_DISPLAY_ON 0b000000000000001110`
- `#define READ_BUSY_FLAG 0b0000000010000000`
- `#define CURSOR_OR_DISPLAY 0b00000000000010100`

Funktionen

- void `initPMP` (void)
Initialisierung des PMPs.
- void `lcd_write_data` (uint8_t `data`)
Schreiben von Daten auf den Speicher des LCDs.
- uint8_t `lcd_get_status` (void)
Liest den aktuellen Status des LCDs aus.
- void `waitForBusyLCD` (void)
Ruft `lcd_get_status()` zyklisch auf, bis Busy Flag nicht mehr gesetzt ist.
- void `writeStrLCD` (const char *`str`)
Gibt einen String an der aktuellen Position im Display aus.
- void `lcd_clear` (void)
Sendet den Befehl, das LCD zurückzusetzen und den Inhalt zu löschen.
- void `lcd_set_pos` (int `line`, int `pos`)
Setzt die Position des Cursors.

10.19.1 Makro-Dokumentation

10.19.1.1 __delay_cycles

```
#define __delay_cycles(
    cycles ) asm volatile ("repeat #%0 \n nop" : : "i" (cycles-2))
```

Definiert in Zeile 21 der Datei [PMP.h](#).

10.19.1.2 CURSOR_OR_DISPLAY

```
#define CURSOR_OR_DISPLAY 0b00000000000010100
```

Definiert in Zeile 61 der Datei [PMP.h](#).

10.19.1.3 LCD_CMD_INIT

```
#define LCD_CMD_INIT 0b00000000000110000 /*LCD wird Initialisiert*/
```

Definiert in Zeile 28 der Datei [PMP.h](#).

10.19.1.4 LCD_DATA

```
#define LCD_DATA(  
    d ) LATE = (LATE & 0xFF00) | d
```

Definiert in Zeile 23 der Datei [PMP.h](#).

10.19.1.5 LCD_DISPLAY_CLEAR

```
#define LCD_DISPLAY_CLEAR 0b00000000000000000001
```

Definiert in Zeile 38 der Datei [PMP.h](#).

10.19.1.6 LCD_DISPLAY_HOME

```
#define LCD_DISPLAY_HOME 0b000000000000000000010
```

Definiert in Zeile 42 der Datei [PMP.h](#).

10.19.1.7 LCD_DISPLAY_OFF

```
#define LCD_DISPLAY_OFF 0b00000000000000000001000
```

Definiert in Zeile 35 der Datei [PMP.h](#).

10.19.1.8 LCD_DISPLAY_ON

```
#define LCD_DISPLAY_ON 0b00000000000000000001110
```

Definiert in Zeile 51 der Datei [PMP.h](#).

10.19.1.9 LCD_ENABLE

```
#define LCD_ENABLE _RD4 /*LCD Aktivierungssignal*/
```

Definiert in Zeile 26 der Datei [PMP.h](#).

10.19.1.10 LCD_ENTRY_MODE

```
#define LCD_ENTRY_MODE 0b0000000000000000110
```

Definiert in Zeile 45 der Datei [PMP.h](#).

10.19.1.11 LCD_FUNCTION_SET

```
#define LCD_FUNCTION_SET 0b00000000000111000
```

Definiert in Zeile 32 der Datei [PMP.h](#).

10.19.1.12 LCD_R_W

```
#define LCD_R_W _RD5 /*LCD Daten Lesen oder Schreiben*/
```

Definiert in Zeile 25 der Datei [PMP.h](#).

10.19.1.13 LCD_RS

```
#define LCD_RS _RB15 /*LCD Registerauswahlsignal*/
```

Definiert in Zeile 24 der Datei [PMP.h](#).

10.19.1.14 READ_BUSY_FLAG

```
#define READ_BUSY_FLAG 0b0000000010000000
```

Definiert in Zeile 55 der Datei [PMP.h](#).

10.19.2 Dokumentation der Funktionen

10.19.2.1 initPMP()

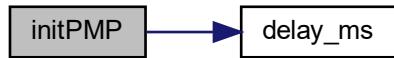
```
void initPMP (
    void )
```

Initialisierung des PMPs.

Drei Function Sets und anschließend die Konfiguration des LCDs. Diese beinhaltet das Funktion Set, Display off, Display clear, Entry mode und Display on.

Definiert in Zeile 29 der Datei [PMP.c](#).

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



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



10.19.2.2 lcd_clear()

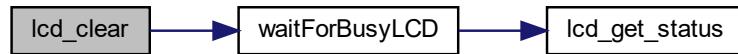
```
void lcd_clear (
    void )
```

Sendet den Befehl, das LCD zurückzusetzen und den Inhalt zu löschen.

Vor dem Zugriff wird Funktion [waitForBusyLCD\(\)](#) solange blockieren, bis Busy Flag nicht gesetzt.

Definiert in Zeile 135 der Datei [lcd_gpio.c](#).

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



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



10.19.2.3 lcd_get_status()

```
uint8_t lcd_get_status (
    void )
```

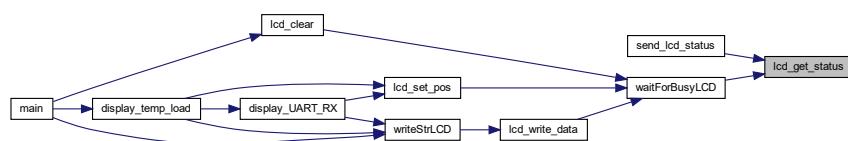
Liest den aktuellen Status des LCDs aus.

Rückgabe

8bit breiter Status, Bit 7 ist Busy Flag
PMDIN1 wird zum Puffer eingehender Daten verwendet

Definiert in Zeile [230](#) der Datei [lcd_gpio.c](#).

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



10.19.2.4 lcd_set_pos()

```
void lcd_set_pos (
    int line,
    int pos )
```

Setzt die Position des Cursors.

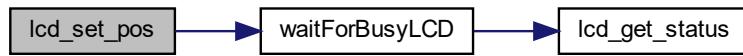
Vor dem Zugriff wird Funktion [waitForBusyLCD\(\)](#) solange blockieren, bis Busy Flag nicht gesetzt.

Parameter

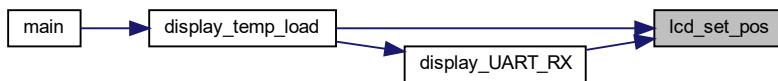
<i>line</i>	Linie, entweder 1 oder 2
<i>pos</i>	Position

Definiert in Zeile [175](#) der Datei [lcd_gpio.c](#).

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



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



10.19.2.5 lcd_write_data()

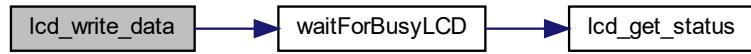
```
void lcd_write_data (
    uint8_t data )
```

Schreiben von Daten auf den Speicher des LCDs.

Vor dem Zugriff wird Funktion [waitForBusyLCD\(\)](#) solange blockieren, bis Busy Flag nicht gesetzt.

Definiert in Zeile [114](#) der Datei [lcd_gpio.c](#).

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



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



10.19.2.6 waitForBusyLCD()

```
void waitForBusyLCD (
    void )
```

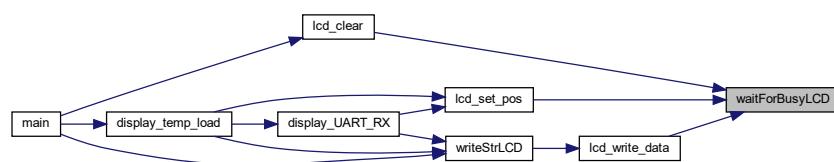
Ruft [lcd_get_status\(\)](#) zyklisch auf, bis Busy Flag nicht mehr gesetzt ist.

Definiert in Zeile 251 der Datei [lcd_gpio.c](#).

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



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



10.19.2.7 writeStrLCD()

```
void writeStrLCD (
    const char * str )
```

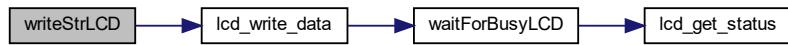
Gibt einen String an der aktuellen Position im Display aus.

Parameter

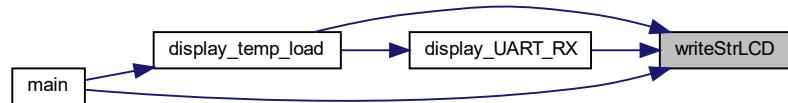
<code>str</code>	Zeichenkette
------------------	--------------

Definiert in Zeile 157 der Datei [lcd_gpio.c](#).

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



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



10.20 PMP.h

[gehe zur Dokumentation dieser Datei](#)

```

00001
00002 /* Files to Include
00003
00005 #include "user.h"          /* Benutzer - Funktion/Parameter
00006
00007 #include <stdint.h>         /* Enthält uint16_t-Definition
00008 #include <stdbool.h>        /* Enthält eine Wahr/Falsch-Definition
00009 #include <string.h>          /* Enthält Zeichenketten
00010 #include <stdio.h>           /* Enthält Ein - und Ausgabefunktionen
00011 #include <stdlib.h>          /* Enthält Hilfsfunktionen
00012
00013 #include <xc.h>             /* Jede Prozessordatei ist geschützt.
00014
00015
00016 /* Konstanten
00017
00019 /*volatile darf nicht vom Compiler wegoptimiert werden und kann extern
00020   *verändert werden z.b. von Interruptroutine oder Timer*/
00021 #define __delay_cycles(cycles)asm volatile ("repeat #%0 \n nop" : : "i" (cycles-2))
00022
00023 #define LCD_DATA(d) LATE = (LATE & 0xFF00) | d
00024 #define LCD_RS _RB15          /*LCD Registerauswahlsignal*/
00025 #define LCD_R_W _RD5          /*LCD Daten Lesen oder Schreiben*/
  
```

```

00026 #define LCD_ENABLE _RD4      /*LCD Aktivierungssignal*/
00027
00028 #define LCD_CMD_INIT        0b00000000000110000 /*LCD wird Initialisiert*/
00029
00030 /*Einstellen der Bewegungsrichtung von Cursor und Display. (8-Bit-Busbetrieb,
00031 *2-zeilige Anzeigemodus eingestellt) Benötigt 38us*/
00032 #define LCD_FUNCTION_SET     0b00000000000111000
00033
00034 /*Schaltet das Display aus. Benötigt 38us*/
00035 #define LCD_DISPLAY_OFF      0b00000000000001000
00036
00037 /*Löschen der Anzeigedaten. Benötigt 1.52ms*/
00038 #define LCD_DISPLAY_CLEAR    0b000000000000000001
00039
00040 /*Cursor und Anzeige werden an ihren ursprünglichen Zustand versetzt.
00041 *Benötigt 1.52ms*/
00042 #define LCD_DISPLAY_HOME     0b000000000000000000010
00043
00044 /*Eingabemodus eingestellt. Cursor bewegt sich nach Rechts. Benötigt 38us*/
00045 #define LCD_ENTRY_MODE       0b0000000000000000000110
00046
00047 /*Schaltet Display und Cursor ein und lässt den Cursor blinken. Benötigt 38us*/
00048 // #define LCD_DISPLAY_ON    0b000000000000000111
00049 /*Schaltet Display und Cursor ein und lässt den Cursor nicht mehr blinken.
00050 *Benötigt 38us*/
00051 #define LCD_DISPLAY_ON       0b000000000000000110
00052
00053 /*Definiert den IC und gibt an das dieser in Betrieb ist. Interne Operation ist
00054 *im Gange es soll gewartet werden. Benötigt 0us.*/
00055 #define READ_BUSY_FLAG       0b000000001000000
00056
00057 /*Cursor nach rechts schieben, AC wird um 1 erhöht und gesamte Anzeige nach
00058 * rechts verschieben (der Cursor bewegt sich entsprechend der Anzeige).
00059 * In der Anzeige wird nicht gelesen oder geschrieben. Diese Anweisung wird
00060 * verwendet, um Anzeigedaten zu korrigieren oder zu suchen. Benötigt 38us. */
00061 #define CURSOR_OR_DISPLAY    0b00000000000010100
00062
00063
00064 /* Prototypen */ *
00065
00066 void initPMP(void);
00067 void lcd_write_data(uint8_t data);
00068 uint8_t lcd_get_status(void);
00069 void waitForBusyLCD(void);
00070 void writeStrLCD(const char* str);
00071 void lcd_clear(void);
00072 void lcd_set_pos(int line, int pos);
00073
00074
00075
00076

```

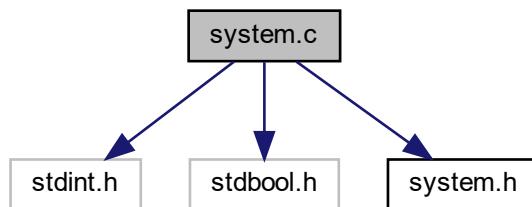
10.21 system.c-Dateireferenz

```

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

```

Include-Abhängigkeitsdiagramm für system.c:



Makrodefinitionen

- `#define CYCLES_PER_MILLISECONDS ((335ull*SYS_FREQ)/7370000ull)`
- `#define CYCLES_PER_MIKROSECONDS CYCLES_PER_MILLISECONDS/1000`

Funktionen

- `void ConfigureOscillator (void)`
- `void init_timer1 ()`
- `void init_ms_t4 ()`
- `void init_t2_t3 ()`
- `void delay_ms (uint16_t milliseconds)`
Delay in ms Blockierende Delay Funktion, eventuell nicht perfekte Verzögerung mit Hilfe einer for Schleife.
- `void delay_us (uint16_t mikoseconds)`

Variablen

- `uint32_t DELAY_ANPASSUNG = ((SYS_FREQ/96)*2180ull)/1000000ull`

10.21.1 Makro-Dokumentation

10.21.1.1 CYCLES_PER_MIKROSECONDS

```
#define CYCLES_PER_MIKROSECONDS CYCLES_PER_MILLISECONDS/1000
```

Definiert in Zeile 17 der Datei [system.c](#).

10.21.1.2 CYCLES_PER_MILLISECONDS

```
#define CYCLES_PER_MILLISECONDS ((335ull*SYS_FREQ)/7370000ull)
```

Definiert in Zeile 16 der Datei [system.c](#).

10.21.2 Dokumentation der Funktionen

10.21.2.1 ConfigureOscillator()

```
void ConfigureOscillator (
    void )
```

Definiert in Zeile 46 der Datei [system.c](#).

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



10.21.2.2 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 156 der Datei [system.c](#).

10.21.2.3 delay_us()

```
void delay_us (
    uint16_t mikroseconds )
```

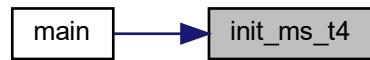
Definiert in Zeile 166 der Datei [system.c](#).

10.21.2.4 init_ms_t4()

```
void init_ms_t4 (
    void )
```

Definiert in Zeile 123 der Datei [system.c](#).

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



10.21.2.5 init_t2_t3()

```
void init_t2_t3 (
    void )
```

Definiert in Zeile 136 der Datei [system.c](#).

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



10.21.2.6 init_timer1()

```
void init_timer1 (
    void )
```

Definiert in Zeile 106 der Datei [system.c](#).

10.21.3 Variablen-Dokumentation

10.21.3.1 DELAY_ANPASSUNG

```
uint32_t DELAY_ANPASSUNG = ((SYS_FREQ/96)*2180ull)/1000000ull
```

Definiert in Zeile 26 der Datei [system.c](#).

10.22 system.c

[gehe zur Dokumentation dieser Datei](#)

```
00001
00002 /* Files to Include
00003
00004 */
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 <p33Fxeee.h>
00013     #endif
00014 #endif
00015
00016 #define CYCLES_PER_MILLISECONDS ((335ull*SYS_FREQ)/7370000ull)
00017 #define CYCLES_PER_MIKROSECONDS CYCLES_PER_MILLISECONDS/1000
00018
00019
00020 #include <stdint.h>          /* For uint16_t definition
00021 #include <stdbool.h>          /* For true/false definition
00022
00023 #include "system.h"          /* variables/params used by system.c
00024
00025
00026 uint32_t DELAY_ANPASSUNG = ((SYS_FREQ/96)*2180ull)/1000000ull; //Berechnung der Delay Anpassung
00027
00028
00029 /* System Level Functions
00030 */
00031 /* Custom oscillator configuration funtions, reset source evaluation
00032 /* functions, and other non-peripheral microcontroller initialization
00033 /* functions get placed in system.c.
00034 */
00035
00036 /* Refer to the device Family Reference Manual Oscillator section for
00037 information about available oscillator configurations. Typically
00038 this would involve configuring the oscillator tuning register or clock
00039 switching useing the compiler's __builtin_write_OSCCON functions.
00040 Refer to the C Compiler for PIC24 MCUs and dsPIC DSCs User Guide in the
00041 compiler installation directory /doc folder for documentation on the
00042 __builtin functions.*/
00043
00044
00045 /* TODO Add clock switching code if appropriate. An example stub is below. */
00046 void ConfigureOscillator(void)
00047 {
00048     //Nur umschalten auf Primary (8 MHz) wenn höhere Frequenz erwünscht
00049     if (SYS_FREQ>7370000L)
00050     {
00051         switch (SYS_FREQ)
00052         {
00053             case 8000000L:
00054                 //PLL muss nicht konfiguriert werden
00055                 // externer Quartz mit 8Mhz
00056                 break;
00057             case 5000000L:
00058                 CLKDIVbits.PLLPOST=2; //N2=4
00059                 PLLFBD=48; //M=50
00060                 CLKDIVbits.PLLPRE=0; //N1=2
00061                 break;
00062             case 7000000L:
00063                 CLKDIVbits.PLLPOST=2; //N2=4
```

```

00064         PLLFBD=188; //M=190
00065         CLKDIVbits.PLLPRE=3; //N1=5
00066         break;
00067     case 100000000L:
00068         CLKDIVbits.PLLPOST=0; //N2=2
00069         PLLFBD=123; //M=125
00070         CLKDIVbits.PLLPRE=3; //N1=5
00071         break;
00072     case 140000000L:
00073         CLKDIVbits.PLLPOST=0; //N2=2
00074         PLLFBD=173; //M=175
00075         CLKDIVbits.PLLPRE=3; //N1=5
00076         break;
00077     //default:
00078     //##error Tets
00079 }
00080 OSCTUN = 0;
00081
00082 if (SYS_FREQ == 8000000L)
00083 {
00084     __builtin_write_OSCCONH(0x02); //Switch auf Primary ohne PLL
00085
00086     __builtin_write_OSCCONL(OSCCON | 0x01);
00087     while (OSCCONbits.COSC!= 0x02); //Warten bis gewechselt wurde
00088 }
00089
00090 else
00091 {
00092     __builtin_write_OSCCONH(0x03); //Switch auf Primary mit PLL
00093
00094     __builtin_write_OSCCONL(OSCCON | 0x01);
00095
00096     while (OSCCONbits.COSC!= 0x3); //Warten bis gewechselt wurde
00097     while (OSCCONbits.LOCK!= 1);
00098 }
00099
00100 }
00101
00102 /*ConfigureOscillator()*/
00103
00104
00105 //Timer1
00106 void init_timer1() //generiert in 1s Rythmus Interrupts
00107 {
00108     __builtin_write_OSCCONL(0b00000011); //SOSC aktivieren
00109     T1CONbits.TON = 0; // Disable Timer
00110     T1CONbits.TCS = 1; // Select external clock
00111     T1CONbits.TSYNC = 0; // Disable Synchronization
00112     T1CONbits.TCKPS = 0b00; // Select 1:1 Prescaler
00113     TMRI = 0x00; // Clear timer register
00114     PR1 = 32767; // Load the period value, Quarztakt
00115
00116     IPC0bits.T1IP = 2; // Set Timer 1 Interrupt Priority Level
00117     IFS0bits.T1IF = 0; // Clear Timer 1 Interrupt Flag
00118     IEC0bits.T1IE = 1; // Enable Timer1 interrupt
00119     T1CONbits.TON = 1; // Start Timer
00120
00121 /*init_timer1()*/
00122
00123 void init_ms_t4() //Interrupt Flag wird jede ms gesetzt
00124 {
00125     T4CONbits.TON = 0; // Stop any 16/32-bit Timer3 operation
00126     T4CONbits.TCS = 0; // Select internal instruction cycle clock
00127
00128     T4CONbits.TGATE = 0; // Disable Gated Timer mode
00129     T4CONbits.TCKPS = 0b10; // Select 1:64 Prescaler
00130     TMR4 = 0x00; // Clear
00131     PR4 = (FCY/64000)-1; // Load 32-bit period value (lsw)
00132     T4CONbits.TON = 1; // Start Timer
00133
00134 /*init_ms_t4()*/
00135
00136 void init_t2_t3()
00137 {
00138     T3CONbits.TON = 0; // Stop any 16-bit Timer3 operation
00139     T2CONbits.TON = 0; // Stop any 16/32-bit Timer3 operation
00140     T2CONbits.T32 = 1; // Enable 32-bit Timer mode
00141     T2CONbits.TCS = 0; // Select internal instruction cycle clock
00142     T2CONbits.TGATE = 1; // Enable Gated Timer mode
00143     T2CONbits.TCKPS = 0b00; // Select 1:1 Prescaler
00144     TMR3 = 0x00; // Clear 32-bit Timer (msw)
00145     TMR2 = 0x00; // Clear 32-bit Timer (lsw)
00146     PR3 = 0xFFFF; // Load 32-bit period value (msw)
00147     PR2 = 0xFFFF; // Load 32-bit period value (lsw)
00148     IPC2bits.T3IP = 0x01; // Set Timer3 Interrupt Priority Level
00149     IFS0bits.T3IF = 0; // Clear Timer3 Interrupt Flag
00150     IEC0bits.T3IE = 0; // Disable Timer3 interrupt

```

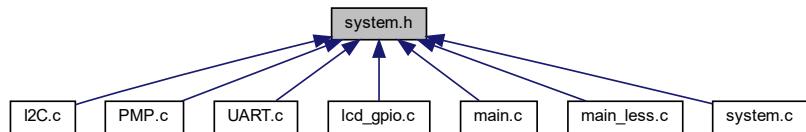
```

00151     T2CONbits.TON = 1; // Start 32-bit Timer
00152
00153 /*init_t2_t3()*/
00154
00155
00156 void delay_ms(uint16_t milliseconds)
00157 {
00158     uint32_t i=0;
00159     for (i=0;i<(DELAY_ANPASSUNG*(uint32_t)milliseconds);i++)
00160     {
00161
00162     }
00163
00164 /*delay_ms()*/
00165
00166 void delay_us(uint16_t mikroseconds)
00167 {
00168     int i, j;
00169     for (i = 0; i < mikroseconds; i++)
00170     {
00171         for(j = 0; j < CYCLES_PER_MIKROSECONDS; j++)
00172         {
00173             Nop();
00174         }
00175     }
00176
00177 /*delay_us()*/

```

10.23 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)
- void **init_t2_t3** (void)

Variablen

- uint32_t **DELAY_ANPASSUNG**

10.23.1 Makro-Dokumentation

10.23.1.1 FCY

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

Definiert in Zeile 15 der Datei [system.h](#).

10.23.1.2 SYS_FREQ

```
#define SYS_FREQ 50000000L
```

Definiert in Zeile 10 der Datei [system.h](#).

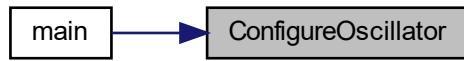
10.23.2 Dokumentation der Funktionen

10.23.2.1 ConfigureOscillator()

```
void ConfigureOscillator (
    void )
```

Definiert in Zeile 46 der Datei [system.c](#).

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



10.23.2.2 delay_ms()

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

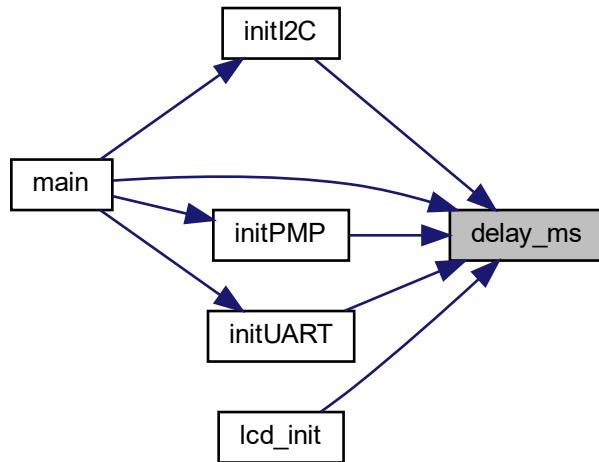
Delay in ms Blockierende Delay Funktion, eventuell nicht perfekte Verzögerung mit Hilfe einer for Schleife.

Parameter

in	<i>milliseconds</i>	Verzögerungszeit in millisekunden
----	---------------------	-----------------------------------

Definiert in Zeile 85 der Datei [main_less.c](#).

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



10.23.2.3 `init_ms_t4()`

```
void init_ms_t4 (
    void )
```

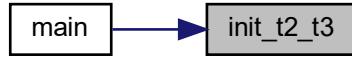
Definiert in Zeile 123 der Datei [system.c](#).

10.23.2.4 `init_t2_t3()`

```
void init_t2_t3 (
    void )
```

Definiert in Zeile 136 der Datei [system.c](#).

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



10.23.2.5 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:



10.23.3 Variablen-Dokumentation

10.23.3.1 DELAY_ANPASSUNG

```
uint32_t DELAY_ANPASSUNG [extern]
```

Definiert in Zeile 39 der Datei [main_less.c](#).

10.24 system.h

[gehe zur Dokumentation dieser Datei](#)

```

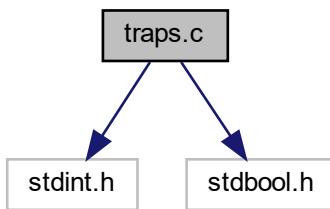
00001
00002 /* System Level #define Macros
00003
00005 /* TODO Define system operating frequency */
00006
00007 /* Microcontroller MIPs (FCY) */
00008 // #define SYS_FREQ      7370000L
00009 // #define SYS_FREQ      8000000L
00010 #define SYS_FREQ      50000000L
00011 // #define SYS_FREQ      70000000L
00012 // #define SYS_FREQ      100000000L
00013 // #define SYS_FREQ      140000000L
00014
00015 #define FCY          SYS_FREQ/2
00016
00017
00018
00019
00020 extern uint32_t DELAY_ANPASSUNG;
00021
00022
00023
00024
00025 /* System Function Prototypes
00026
00028 /* Custom oscillator configuration funtions, reset source evaluation
00029 functions, and other non-peripheral microcontroller initialization functions
00030 go here. */
00031
00032
00033 //System Prototypen
00034 void ConfigureOscillator(void); /* Handles clock switching/osc initialization */
00035 void delay_ms(uint16_t milliseconds);
00036
00037 void init_timer1(void);
00038 void init_ms_t4(void);
00039
00040 void init_t2_t3(void);
00041

```

10.25 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)

10.25.1 Dokumentation der Funktionen

10.25.1.1 [_AddressError\(\)](#)

```
void _AddressError (
    void )
```

Definiert in Zeile [82](#) der Datei [traps.c](#).

10.25.1.2 [_DefaultInterrupt\(\)](#)

```
void _DefaultInterrupt (
    void )
```

Definiert in Zeile [154](#) der Datei [traps.c](#).

10.25.1.3 [_MathError\(\)](#)

```
void _MathError (
    void )
```

Definiert in Zeile [93](#) der Datei [traps.c](#).

10.25.1.4 [_OscillatorFail\(\)](#)

```
void _OscillatorFail (
    void )
```

Definiert in Zeile [76](#) der Datei [traps.c](#).

10.25.1.5 _StackError()

```
void _StackError (
    void )
```

Definiert in Zeile 87 der Datei traps.c.

10.26 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

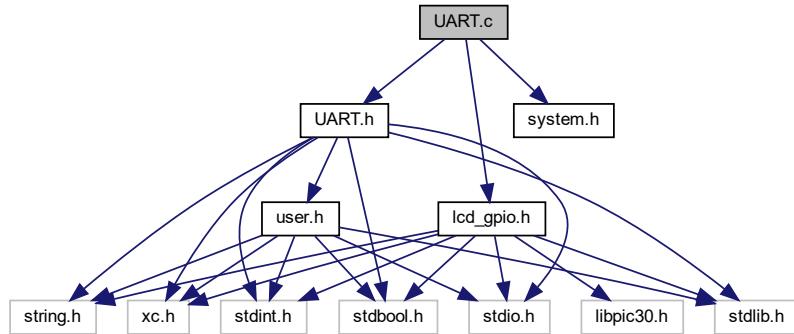
```

10.27 UART.c-Dateireferenz

```

#include "UART.h"
#include "system.h"
#include "lcd_gpio.h"
Include-Abhängigkeitsdiagramm für UART.c:

```



Funktionen

- void [initUART \(\)](#)
- void [_U1TXInterrupt \(void\)](#)
- void [_U1RXInterrupt \(void\)](#)
- int16_t [putcFIFO_TX \(char c\)](#)
- int16_t [getcFIFO_TX \(volatile uint16_t *c\)](#)
- int16_t [putcUART \(char c\)](#)
- int16_t [putsUART \(const char *str\)](#)
- int16_t [putcFIFO_RX \(char c\)](#)
- int16_t [getcFIFO_RX \(char *c\)](#)

Variablen

- Buffer FIFO = {}, 0, 0}
- Buffer FIFO_RX = {}, 0, 0}

10.27.1 Dokumentation der Funktionen

10.27.1.1 _U1RXInterrupt()

```
void _U1RXInterrupt (
    void )
```

Definiert in Zeile 61 der Datei [UART.c](#).

10.27.1.2 _U1TXInterrupt()

```
void _U1TXInterrupt (
    void )
```

Definiert in Zeile 54 der Datei [UART.c](#).

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



10.27.1.3 getcFIFO_RX()

```
int16_t getcFIFO_RX (
    char * c )
```

Definiert in Zeile 183 der Datei [UART.c](#).

10.27.1.4 getcFIFO_TX()

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

Definiert in Zeile 107 der Datei [UART.c](#).

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



10.27.1.5 initUART()

```
void initUART (
    void )
```

Definiert in Zeile 16 der Datei [UART.c](#).

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



10.27.1.6 putcFIFO_RX()

```
int16_t putcFIFO_RX (
    char c )
```

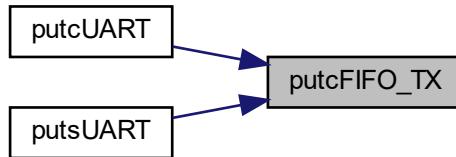
Definiert in Zeile 158 der Datei [UART.c](#).

10.27.1.7 putcFIFO_TX()

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

Definiert in Zeile 82 der Datei [UART.c](#).

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



10.27.1.8 putcUART()

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

Definiert in Zeile 127 der Datei [UART.c](#).

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



10.27.1.9 putsUART()

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

Definiert in Zeile 137 der Datei [UART.c](#).

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



10.27.2 Variablen-Dokumentation

10.27.2.1 FIFO

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

Definiert in Zeile 10 der Datei [UART.c](#).

10.27.2.2 FIFO_RX

```
Buffer FIFO_RX = {{}, 0, 0}
```

Definiert in Zeile 11 der Datei [UART.c](#).

10.28 UART.c

[gehe zur Dokumentation dieser Datei](#)

```

00001
00002 /* Files to Include */ */
00003
00005 #include "UART.h"
00006 #include "system.h"
00007 #include "lcd_gpio.h"
00008
00009
00010 Buffer FIFO = {{}, 0, 0}; //FIFO zum Versenden über UART
00011 Buffer FIFO_RX = {{}, 0, 0}; //FIFO zum Empfangen über UART
00012
00013 /* Funktionen */ */
00014
00016 void initUART()
```

```

00017 {
00018     U1MODEbits.STSEL = 0; // 1-Stop bit
00019     U1MODEbits.PDSEL = 0; // No Parity, 8-Data bits
00020     U1MODEbits.ABAUD = 0; // Auto-Baud disabled
00021     U1MODEbits.UEN = 0;
00022     U1MODEbits.LPBACK = 0;
00023     U1MODEbits.RXINV = 0;
00024     //U1MODEbits.ALTIO = 0;
00025
00026     U1MODEbits.URXINV = 0;
00027     U1MODEbits.RTSMD = 0;
00028
00029     U1MODEbits.BRGH = 0; // Standard-Speed mode
00030     U1BRG = BRGVAL; // Baud Rate setting for 9600
00031
00032     U1STAbits.UTXISEL0 = 0; // Interrupt after one TX character is transmitted
00033     U1STAbits.UTXISEL1 = 0;
00034     U1STAbits.UTXBKR = 0;
00035     U1STAbits.ADDEN = 0;
00036     U1STAbits.UTXINV = 0;
00037     U1STAbits.URXISEL = 0;
00038     U1STA = U1STA | 0b0001000000000000;
00039     //_UIRXEN = 1;
00040
00041     _U1RXIE = 1; // Enable UART RX interrupt
00042
00043     U1MODEbits.UARTEN = 1; // Enable UART
00044     delay_ms(2);
00045     U1STAbits.UTXEN = 1; // Enable UART TX
00046
00047     /* Wait at least 105 microseconds (1/9600) before sending first char */
00048     delay_ms(2);
00049     _U1TXIE = 1; // Enable UART TX interrupt
00050
00051 } /* initUART() */
00052
00053
00054 void __attribute__((__interrupt__, no_auto_psv)) _U1TXInterrupt(void)
00055 {
00056     _U1TXIF = 0; // Clear TX Interrupt flag
00057    getcFIFO_RX(&U1TXREG);
00058 }
00059
00060
00061 void __attribute__((__interrupt__, no_auto_psv)) _U1RXInterrupt(void)
00062 {
00063     _U1RXIF = 0;
00064     char dummy;
00065     static int aufrufe = 0;
00066     if (aufrufe != 0)
00067     {
00068         //putcFIFO_RX(U1RXREG);
00069         received_UART[UART_RX_count]=U1RXREG;
00070         received_UART[UART_RX_count+1]='\0';
00071         UART_RX_count++;
00072     }
00073 }
00074
00075 else
00076 {
00077     dummy = U1RXREG;
00078 }
00079 aufrufe = 1;
00080 }
00081
00082 int16_t putcFIFO_TX(char c)
00083 {
00084     _GIE=0;
00085     //if (buffer.write >= BUFFER_SIZE)
00086     //    buffer.write = 0; // erhöht sicherheit
00087     _LATFO = 1;
00088     if ( ( FIFO.write + 1 == FIFO.read ) ||
00089         ( FIFO.read == 0 && FIFO.write + 1 == BUFFER_SIZE ) )
00090     {
00091         _GIE=1;
00092         return BUFFER_FAIL; // voll
00093     }
00094
00095     FIFO.data[FIFO.write] = c;
00096
00097     FIFO.write++;
00098     if (FIFO.write >= BUFFER_SIZE)
00099     {
00100         FIFO.write = 0;
00101     }
00102     _GIE=1;
00103     return BUFFER_SUCCESS;

```

```

00104 } /* putcFIFO_TX() */
00105
00106
00107 int16_t getcFIFO_TX(volatile uint16_t *c)
00108 {
00109     _GIE=0;
00110     if (FIFO.read == FIFO.write)
00111     {
00112         _GIE=1;
00113         return BUFFER_FAIL;
00114     }
00115     *c = FIFO.data[FIFO.read];
00116
00117     FIFO.read++;
00118     if (FIFO.read >= BUFFER_SIZE)
00119     {
00120         FIFO.read = 0;
00121     }
00122     _GIE=1;
00123     return BUFFER_SUCCESS;
00124
00125 } /* getcFIFO_TX() */
00126
00127 int16_t putcUART(char c)
00128 {
00129     _LATF0 = 1;
00130     _GIE = 0; // Interrupts ausschalten
00131     int16_t erfolg = putcFIFO_TX(c);
00132     _GIE = 1;
00133     return erfolg;
00134
00135 } /* putcUART() */
00136
00137 int16_t putsUART(const char *str)
00138 {
00139     uint16_t i;
00140     uint16_t length = strlen(str);
00141
00142     _GIE = 0; // Global Interrupt disable
00143     for(i = 0; i < length; i++)
00144     {
00145         if(! putcFIFO_TX(str[i]))
00146             break;
00147     }
00148     _GIE = 1;
00149     int16_t erfolg = -i;
00150     if(erfolg == -length)
00151         erfolg *= -1;
00152     _U1TXIF = 1; // Interrupt Routine Starten um FIFO-Inhalt zu senden
00153     return erfolg;
00154
00155 } /* putsUART() */
00156
00157
00158 int16_t putcFIFO_RX(char c)
00159 {
00160     _GIE=0;
00161     //if (buffer.write >= BUFFER_SIZE)
00162     //    buffer.write = 0; // erhöht sicherheit
00163     if ( ( FIFO_RX.write + 1 == FIFO_RX.read ) ||
00164         ( FIFO_RX.read == 0 && FIFO_RX.write + 1 == BUFFER_SIZE ) )
00165     {
00166         return BUFFER_FAIL; // voll
00167         _GIE=1;
00168     }
00169
00170
00171     FIFO_RX.data[FIFO_RX.write] = c;
00172
00173     FIFO_RX.write++;
00174     if (FIFO_RX.write >= BUFFER_SIZE)
00175     {
00176         FIFO_RX.write = 0;
00177     }
00178     _GIE=1;
00179     return BUFFER_SUCCESS;
00180 }
00181
00182
00183 int16_t getcFIFO_RX(char *c)
00184 {
00185     _GIE=0;
00186     if (FIFO_RX.read == FIFO_RX.write)
00187     {
00188         _GIE=1;
00189         return BUFFER_FAIL;
00190     }

```

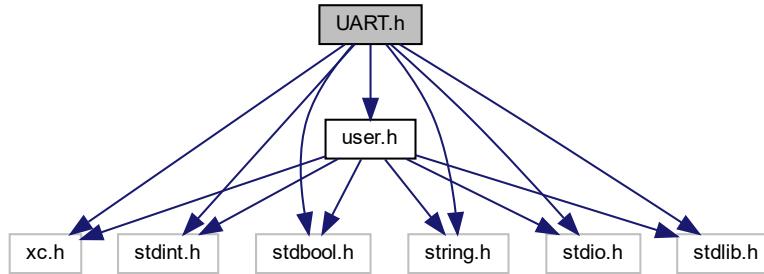
```

00191     *c = FIFO_RX.data[FIFO_RX.read];
00192
00193     FIFO_RX.read++;
00194     if (FIFO_RX.read >= BUFFER_SIZE)
00195     {
00196         FIFO_RX.read = 0;
00197     }
00198     _GIE=1;
00199     return BUFFER_SUCCESS;
00200 }
00201 }
```

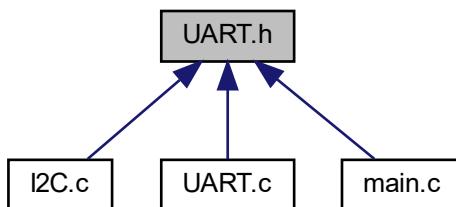
10.29 UART.h-Dateireferenz

```
#include <xc.h>
#include <stdint.h>
#include <stdbool.h>
#include <string.h>
#include <stdio.h>
#include <stdlib.h>
#include "user.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 getcFIFO_RX (char *c)`
- `int16_t putcFIFO_TX (char c)`
- `int16_t putcFIFO_RX (char c)`

Variablen

- `Buffer FIFO`
- `Buffer FIFO_RX`
- `char received_UART [20]`
- `int UART_RX_count`

10.29.1 Makro-Dokumentation

10.29.1.1 BAUDRATE

```
#define BAUDRATE 9600
```

Definiert in Zeile 19 der Datei [UART.h](#).

10.29.1.2 BRGVAL

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

Definiert in Zeile 20 der Datei [UART.h](#).

10.29.2 Dokumentation der Funktionen

10.29.2.1 getcFIFO_RX()

```
int16_t getcFIFO_RX (  
    char * c )
```

Definiert in Zeile 183 der Datei [UART.c](#).

10.29.2.2 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:



10.29.2.3 initUART()

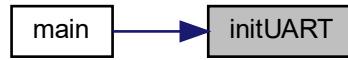
```
void initUART (  
    void )
```

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:



10.29.2.4 putcFIFO_RX()

```
int16_t putcFIFO_RX (
    char c )
```

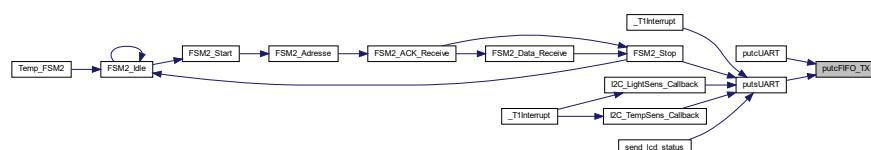
Definiert in Zeile 158 der Datei [UART.c](#).

10.29.2.5 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:



10.29.2.6 putsUART()

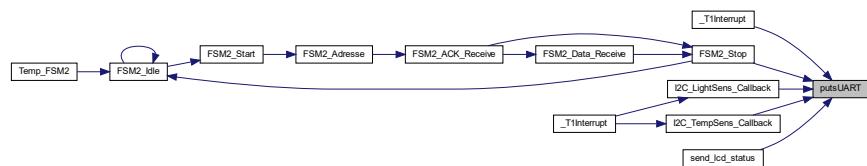
```
int16_t putsUART (
    const char * str )
```

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:



10.29.3 Variablen-Dokumentation

10.29.3.1 FIFO

`Buffer FIFO [extern]`

Definiert in Zeile 56 der Datei [main_less.c](#).

10.29.3.2 FIFO_RX

`Buffer FIFO_RX [extern]`

Definiert in Zeile 11 der Datei [UART.c](#).

10.29.3.3 received_UART

```
char received_UART[20] [extern]
```

Definiert in Zeile 49 der Datei [main.c](#).

10.29.3.4 UART_RX_count

```
int UART_RX_count [extern]
```

Definiert in Zeile 50 der Datei [main.c](#).

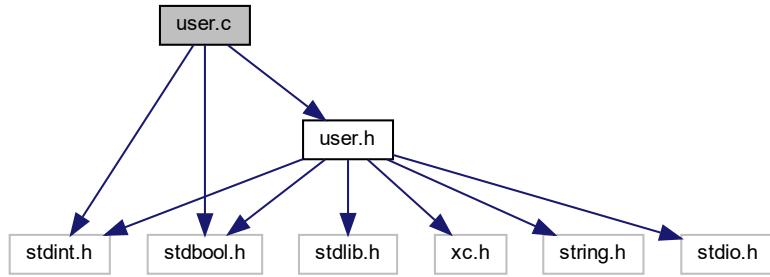
10.30 UART.h

[gehe zur Dokumentation dieser Datei](#)

```
00001
00002 /* Files to Include */ 
00003
00005 #include <xc.h> /* Jede Prozessordatei ist geschützt. */
00006
00007 #include <stdint.h> /* Enthält uint16_t-Definition */
00008 #include <stdbool.h> /* Enthält eine Wahr/Falsch-Definition */
00009 #include <string.h> /* Enthält Zeichenketten */
00010 #include <stdio.h> /* Enthält Ein - und Ausgabefunktionen */
00011 #include <stdlib.h> /* Enthält Hilfsfunktionen */
00012
00013 #include "user.h" /* Benutzer - Funktion/Parameter */
00014
00015
00016 /* Konstanten */ 
00017
00019 #define BAUDRATE 9600
00020 #define BRGVAL ((FCY/BAUDRATE)/16)-1
00021
00022
00023 /* Typedef */ 
00024
00026 typedef struct
00027 {
00028     uint8_t data[BUFFER_SIZE];
00029     uint8_t read; // zeigt auf das Feld mit dem ältesten Inhalt
00030     uint8_t write; // zeigt immer auf leerer Feld
00031 }Buffer;
00032
00033
00034 /* Globale Variable Declaration */ 
00035
00037 extern Buffer FIFO;
00038 extern Buffer FIFO_RX;
00039 extern char received_UART[20];
00040 extern int UART_RX_count;
00041
00042
00043 /* Prototypen */ 
00044
00046 void initUART(void);
00047 int16_t putsUART(const char *str);
00048 int16_t getcFIFO_TX(volatile uint16_t *c);
00049 int16_t getcFIFO_RX(char *c);
00050 int16_t putcFIFO_TX(char c);
00051 int16_t putcFIFO_RX(char c);
```

10.31 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)

10.31.1 Dokumentation der Funktionen

10.31.1.1 [InitApp\(\)](#)

```
void InitApp (
    void )
```

Definiert in Zeile [26](#) der Datei [user.c](#).

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



10.31.1.2 setLED()

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

Definiert in Zeile 35 der Datei [user.c](#).

10.32 user.c

[gehe zur Dokumentation dieser Datei](#)

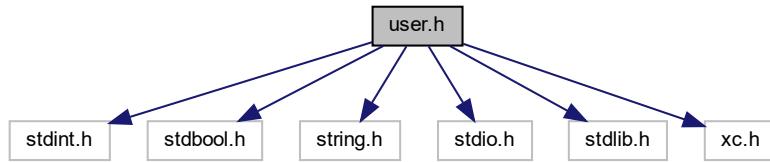
```
00001
00002 /* Files to Include
00003
00005 /* Device header file */
00006 #if defined(__XC16__)
00007     #include <xc.h>
00008 #elif defined(__C30__)
00009     #if defined(__dsPIC33E__)
00010         #include <p33Exxxx.h>
00011     #elif defined(__dsPIC33F__)
00012         #include <p33Fxeee.h>
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);
```

10.33 user.h-Dateireferenz

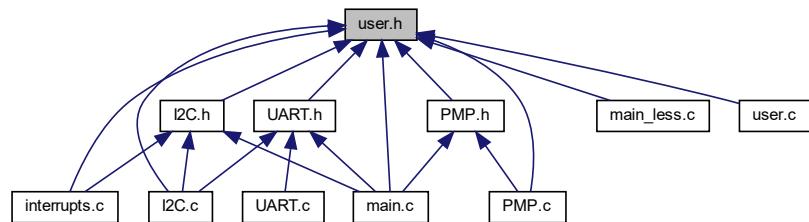
```
#include <stdint.h>
#include <stdbool.h>
#include <string.h>
#include <stdio.h>
#include <stdlib.h>
```

```
#include <xc.h>
```

Include-Abhängigkeitsdiagramm für user.h:



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 1

Funktionen

- void InitApp (void)

10.33.1 Makro-Dokumentation

10.33.1.1 BUFFER_FAIL

```
#define BUFFER_FAIL 0
```

Definiert in Zeile 19 der Datei [user.h](#).

10.33.1.2 BUFFER_SIZE

```
#define BUFFER_SIZE 256
```

Definiert in Zeile 21 der Datei [user.h](#).

10.33.1.3 BUFFER_SUCCESS

```
#define BUFFER_SUCCESS 1
```

Definiert in Zeile 20 der Datei [user.h](#).

10.33.1.4 LED0

```
#define LED0 _LATB8
```

Definiert in Zeile 10 der Datei [user.h](#).

10.33.1.5 LED1

```
#define LED1 _LATB9
```

Definiert in Zeile 11 der Datei [user.h](#).

10.33.1.6 LED2

```
#define LED2 _LATB10
```

Definiert in Zeile 12 der Datei [user.h](#).

10.33.1.7 LED3

```
#define LED3 _LATB11
```

Definiert in Zeile 13 der Datei [user.h](#).

10.33.1.8 SENSOR_TIME

```
#define SENSOR_TIME 1
```

Definiert in Zeile 22 der Datei [user.h](#).

10.33.1.9 T0

```
#define T0 !_RG12
```

Definiert in Zeile 14 der Datei [user.h](#).

10.33.1.10 T1

```
#define T1 !_RG13
```

Definiert in Zeile 15 der Datei [user.h](#).

10.33.1.11 T2

```
#define T2 !_RG14
```

Definiert in Zeile 16 der Datei [user.h](#).

10.33.1.12 T3

```
#define T3 !_RG15
```

Definiert in Zeile 17 der Datei [user.h](#).

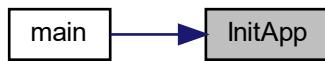
10.33.2 Dokumentation der Funktionen

10.33.2.1 InitApp()

```
void InitApp (
    void )
```

Definiert in Zeile 26 der Datei [user.c](#).

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



10.34 user.h

[gehe zur Dokumentation dieser Datei](#)

```

00001 #include <stdint.h>          /* Includes uint16_t definition
00002 #include <stdbool.h>          /* Includes true/false definition
00003 #include <string.h>
00004 #include <stdio.h>
00005 #include <stdlib.h>
00006 #include <xc.h>
00007
00008 /* User Level #define Macros */ */
00009
00010 #define LED0 _LATB8
00011 #define LED1 _LATB9
00012 #define LED2 _LATB10
00013 #define LED3 _LATB11
00014 #define T0 !_RG12
00015 #define T1 !_RG13
00016 #define T2 !_RG14
00017 #define T3 !_RG15
00018
00019 #define BUFFER_FAIL      0
00020 #define BUFFER_SUCCESS   1
00021 #define BUFFER_SIZE 256
00022 #define SENSOR_TIME 1
00023 /* TODO Application specific user parameters used in user.c may go here */
00024
00025
00026
00027
00028 /* User Function Prototypes */ */
00029
00030 /* TODO User level functions prototypes (i.e. InitApp) go here */
00031
00032
00033 void InitApp(void);           /* I/O and Peripheral Initialization */
00034
00035
  
```

10.35 Code Style.markdown-Dateireferenz

10.36 Dokumentation.markdown-Dateireferenz

Index

_AddressError
 traps.c, 133

_DefaultInterrupt
 traps.c, 133

_MathError
 traps.c, 133

_OscillatorFall
 traps.c, 133

_StackError
 traps.c, 133

_T1Interrupt
 interrupts.c, 60
 main_less.c, 89

_U1RXInterrupt
 UART.c, 137

_U1TXInterrupt
 main_less.c, 90
 UART.c, 137

__delay_cycles
 lcd_gpio.h, 71
 PMP.h, 114

address
 I2C_struct, 21

BAUDRATE
 main_less.c, 87
 UART.h, 144

BRGVAL
 main_less.c, 88
 UART.h, 144

Buffer, 17
 data, 17
 read, 18
 write, 18

BUFFER_FAIL
 main_less.c, 88
 user.h, 151

Buffer_I2C_FSM, 18
 data, 19
 read, 19
 write, 20

BUFFER_SIZE
 main_less.c, 88
 user.h, 152

BUFFER_SUCCESS
 main_less.c, 88
 user.h, 152

callback

I2C_struct, 21

Code Style.markdown, 154

configuration_bits.c, 23

ConfigureOscillator
 system.c, 123
 system.h, 129

CURSOR_OR_DISPLAY
 lcd_gpio.h, 71
 PMP.h, 114

CYCLES_PER_MIKROSECONDS
 system.c, 123

CYCLES_PER_MILLISECONDS
 system.c, 123

data
 Buffer, 17
 Buffer_I2C_FSM, 19
 main_less.c, 99

DELAY_ANPASSUNG
 main_less.c, 99
 system.c, 126
 system.h, 131

delay_ms
 main_less.c, 90
 system.c, 124
 system.h, 129

delay_us
 system.c, 124

display_temp_load
 main.c, 78

display_UART_RX
 main.c, 78

dol2C
 I2C.c, 26
 I2C.h, 47

Dokumentation.markdown, 154

Error
 I2C.h, 46

exchangeI2C
 I2C.c, 26
 I2C.h, 47

FCY
 system.h, 129

FIFO
 main_less.c, 100
 UART.c, 140
 UART.h, 147

FIFO_I2C

I2C.c, 38
 I2C.h, 57
FIFO_RX
 UART.c, 140
 UART.h, 147
Finished
 I2C.h, 46
FSM2_ACK_Receive
 main_less.c, 91
FSM2_Adresse
 main_less.c, 92
FSM2_Data_Receive
 main_less.c, 92
FSM2_Idle
 main_less.c, 93
FSM2_Start
 main_less.c, 93
FSM2_Stop
 main_less.c, 94
FSM_Adresse_Read
 I2C.c, 27
 I2C.h, 48
FSM_Adresse_Write
 I2C.c, 28
 I2C.h, 49
FSM_Idle
 I2C.c, 29
 I2C.h, 50
FSM_RECV_EN
 I2C.c, 30
 I2C.h, 51
FSM_Repeated_Start
 I2C.c, 31
 I2C.h, 52
FSM_Start
 I2C.c, 32
 I2C.h, 53
FSM_Stop
 I2C.c, 32
 I2C.h, 53
get_I2C_struct_FIFO
 I2C.c, 33
get_Light
 main.c, 79
get_Temperatur
 main.c, 79
getcFIFO_RX
 UART.c, 137
 UART.h, 144
getcFIFO_TX
 main_less.c, 94
 UART.c, 137
 UART.h, 145
HEARTBEAT_MS
 main.c, 78
 main_less.c, 88
 I2C.c, 24, 39
 doI2C, 26
 exchangeI2C, 26
FIFO_I2C, 38
FSM_Adresse_Read, 27
FSM_Adresse_Write, 28
FSM_Idle, 29
FSM_RECV_EN, 30
FSM_Repeated_Start, 31
FSM_Start, 32
FSM_Stop, 32
 get_I2C_struct_FIFO, 33
I2C_LightSens_Callback, 34
I2C_TempSens_Callback, 35
I2C_test_struct, 38
 initI2C, 36
 put_I2C_struct_FIFO, 36
I2C.h, 43, 59
 doI2C, 47
Error, 46
 exchangeI2C, 47
FIFO_I2C, 57
Finished, 46
FSM_Adresse_Read, 48
FSM_Adresse_Write, 49
FSM_Idle, 50
FSM_RECV_EN, 51
FSM_Repeated_Start, 52
FSM_Start, 53
FSM_Stop, 53
I2C_Callback_t, 46
I2C_LightSens_Callback, 54
I2C_SCL, 45
I2C_SCL_TRIS, 45
I2C_SDA, 45
I2C_SDA_TRIS, 46
i2c_status_t, 46
I2C_TempSens_Callback, 55
I2C_test_struct, 57
 initI2C, 56
 latest_temperatur, 57
Pending, 46
read_data_buffer_light, 57
read_data_buffer_temp, 57
StateFunc, 46
status_licht, 58
status_temperatur, 58
write_data_buffer_light, 58
write_data_buffer_temp, 58
I2C_Callback_t
 I2C.h, 46
I2C_LightSens_Callback
 I2C.c, 34
 I2C.h, 54
I2C_SCL
 I2C.h, 45
 main_less.c, 88
I2C_SCL_TRIS

I2C.h, 45
main_less.c, 89
I2C_SDA
 I2C.h, 45
 main_less.c, 89
I2C_SDA_TRIS
 I2C.h, 46
 main_less.c, 89
i2c_status_t
 I2C.h, 46
I2C_struct, 20
 address, 21
 callback, 21
 ID, 21
 num_read, 21
 num_write, 21
 readbuf, 22
 status, 22
 writebuf, 22
I2C_TempSens_Callback
 I2C.c, 35
 I2C.h, 55
I2C_test_struct
 I2C.c, 38
 I2C.h, 57
ID
 I2C_struct, 21
init_ms_t4
 main_less.c, 95
 system.c, 124
 system.h, 130
init_t2_t3
 system.c, 125
 system.h, 130
init_timer1
 main_less.c, 95
 system.c, 125
 system.h, 131
InitApp
 user.c, 149
 user.h, 154
initI2C
 I2C.c, 36
 I2C.h, 56
 main_less.c, 96
initPMP
 PMP.c, 106
 PMP.h, 116
initUART
 main_less.c, 96
 UART.c, 138
 UART.h, 145
interrupts.c, 60, 61
 _T1Interrupt, 60
latest_cpu_load
 main.c, 81
latest_temperatur
 I2C.h, 57
 main.c, 81
 lcd_clear
 lcd_gpio.c, 62
 lcd_gpio.h, 74
 PMP.c, 106
 PMP.h, 117
LCD_CMD_INIT
 lcd_gpio.h, 71
 PMP.h, 114
LCD_DATA
 lcd_gpio.h, 72
 PMP.h, 114
LCD_DISPLAY_CLEAR
 lcd_gpio.h, 72
 PMP.h, 115
LCD_DISPLAY_HOME
 lcd_gpio.h, 72
 PMP.h, 115
LCD_DISPLAY_OFF
 lcd_gpio.h, 72
 PMP.h, 115
LCD_DISPLAY_ON
 lcd_gpio.h, 72
 PMP.h, 115
LCD_ENABLE
 lcd_gpio.h, 72
 PMP.h, 115
LCD_ENTRY_MODE
 lcd_gpio.h, 73
 PMP.h, 115
LCD_FUNCTION_SET
 lcd_gpio.h, 73
 PMP.h, 116
lcd_get_status
 lcd_gpio.c, 63
 lcd_gpio.h, 74
 PMP.c, 107
 PMP.h, 118
lcd_gpio.c, 62, 67
 lcd_clear, 62
 lcd_get_status, 63
 lcd_init, 63
 lcd_set_pos, 64
 lcd_write_data, 65
 waitForBusyLCD, 65
 writeStrLCD, 66
lcd_gpio.h, 70, 76
 __delay_cycles, 71
CURSOR_OR_DISPLAY, 71
lcd_clear, 74
LCD_CMD_INIT, 71
LCD_DATA, 72
LCD_DISPLAY_CLEAR, 72
LCD_DISPLAY_HOME, 72
LCD_DISPLAY_OFF, 72
LCD_DISPLAY_ON, 72
LCD_ENABLE, 72
LCD_ENTRY_MODE, 73

LCD_FUNCTION_SET, 73
 lcd_get_status, 74
 lcd_init, 74
 LCD_R_W, 73
 LCD_RS, 73
 lcd_set_pos, 74
 lcd_write_data, 75
 READ_BUSY_FLAG, 73
 waitForBusyLCD, 75
 writeStrLCD, 75
 lcd_init
 lcd_gpio.c, 63
 lcd_gpio.h, 74
 LCD_R_W
 lcd_gpio.h, 73
 PMP.h, 116
 LCD_RS
 lcd_gpio.h, 73
 PMP.h, 116
 lcd_set_pos
 lcd_gpio.c, 64
 lcd_gpio.h, 74
 PMP.c, 107
 PMP.h, 118
 lcd_write_data
 lcd_gpio.c, 65
 lcd_gpio.h, 75
 PMP.c, 108
 PMP.h, 119
 LED0
 user.h, 152
 LED1
 user.h, 152
 LED2
 user.h, 152
 LED3
 user.h, 152
 MAIN
 main.c, 78
 main
 main.c, 79
 main_less.c, 97
 main.c, 77, 83
 display_temp_load, 78
 display_UART_RX, 78
 get_Light, 79
 get_Temperatur, 79
 HEARTBEAT_MS, 78
 latest_cpu_load, 81
 latest_temperatur, 81
 MAIN, 78
 main, 79
 measureProcesstime, 80
 print_sensor_values, 80
 read_data_buffer_light, 81
 read_data_buffer_temp, 81
 received_UART, 82
 send_lcd_status, 80
 status_licht, 82
 status_temperatur, 82
 UART_RX_count, 82
 write_data_buffer_light, 82
 write_data_buffer_temp, 82
 main_less.c, 86, 100
 _T1Interrupt, 89
 _U1TXInterrupt, 90
 BAUDRATE, 87
 BRGVAL, 88
 BUFFER_FAIL, 88
 BUFFER_SIZE, 88
 BUFFER_SUCCESS, 88
 data, 99
 DELAY_ANPASSUNG, 99
 delay_ms, 90
 FIFO, 100
 FSM2_ACK_Receive, 91
 FSM2_Adresse, 92
 FSM2_Data_Receive, 92
 FSM2_Idle, 93
 FSM2_Start, 93
 FSM2_Stop, 94
 getcFIFO_TX, 94
 HEARTBEAT_MS, 88
 I2C_SCL, 88
 I2C_SCL_TRIS, 89
 I2C_SDA, 89
 I2C_SDA_TRIS, 89
 init_ms_t4, 95
 init_timer1, 95
 initI2C, 96
 initUART, 96
 main, 97
 putcFIFO_TX, 97
 putcUART, 98
 putsUART, 98
 StateFunc, 89
 Temp_FSM2, 99
 measureProcesstime
 main.c, 80
 num_read
 I2C_struct, 21
 num_write
 I2C_struct, 21
 Pending
 I2C.h, 46
 PMP.c, 105, 110
 initPMP, 106
 lcd_clear, 106
 lcd_get_status, 107
 lcd_set_pos, 107
 lcd_write_data, 108
 waitForBusyLCD, 108
 writeStrLCD, 109
 PMP.h, 112, 121
 __delay_cycles, 114

CURSOR_OR_DISPLAY, 114
initPMP, 116
lcd_clear, 117
LCD_CMD_INIT, 114
LCD_DATA, 114
LCD_DISPLAY_CLEAR, 115
LCD_DISPLAY_HOME, 115
LCD_DISPLAY_OFF, 115
LCD_DISPLAY_ON, 115
LCD_ENABLE, 115
LCD_ENTRY_MODE, 115
LCD_FUNCTION_SET, 116
lcd_get_status, 118
LCD_R_W, 116
LCD_RS, 116
lcd_set_pos, 118
lcd_write_data, 119
READ_BUSY_FLAG, 116
waitForBusyLCD, 120
writeStrLCD, 120
print_sensor_values
 main.c, 80
put_I2C_struct_FIFO
 I2C.c, 36
putcFIFO_RX
 UART.c, 138
 UART.h, 146
putcFIFO_TX
 main_less.c, 97
 UART.c, 138
 UART.h, 146
putcUART
 main_less.c, 98
 UART.c, 139
putsUART
 main_less.c, 98
 UART.c, 139
 UART.h, 146

read
 Buffer, 18
 Buffer_I2C_FSM, 19
READ_BUSY_FLAG
 lcd_gpio.h, 73
 PMP.h, 116
read_data_buffer_light
 I2C.h, 57
 main.c, 81
read_data_buffer_temp
 I2C.h, 57
 main.c, 81
readbuf
 I2C_struct, 22
received_UART
 main.c, 82
 UART.h, 147

send_lcd_status
 main.c, 80

SENSOR_TIME
 user.h, 153
setLED
 user.c, 149
StateFunc
 I2C.h, 46
 main_less.c, 89
status
 I2C_struct, 22
status_licht
 I2C.h, 58
 main.c, 82
status_temperatur
 I2C.h, 58
 main.c, 82
SYS_FREQ
 system.h, 129
system.c, 122, 126
 ConfigureOscillator, 123
 CYCLES_PER_MIKROSECONDS, 123
 CYCLES_PER_MILLISECONDS, 123
 DELAY_ANPASSUNG, 126
 delay_ms, 124
 delay_us, 124
 init_ms_t4, 124
 init_t2_t3, 125
 init_timer1, 125
system.h, 128, 132
 ConfigureOscillator, 129
 DELAY_ANPASSUNG, 131
 delay_ms, 129
 FCY, 129
 init_ms_t4, 130
 init_t2_t3, 130
 init_timer1, 131
 SYS_FREQ, 129

T0
 user.h, 153
T1
 user.h, 153
T2
 user.h, 153
T3
 user.h, 153
Temp_FSM2
 main_less.c, 99
traps.c, 132, 134
 _AddressError, 133
 _DefaultInterrupt, 133
 _MathError, 133
 _OscillatorFail, 133
 _StackError, 133

UART.c, 136, 140
 _U1RXInterrupt, 137
 _U1TXInterrupt, 137
FIFO, 140
FIFO_RX, 140

getcFIFO_RX, 137
getcFIFO_TX, 137
initUART, 138
putcFIFO_RX, 138
putcFIFO_TX, 138
putcUART, 139
putsUART, 139
UART.h, 143, 148
BAUDRATE, 144
BRGVAL, 144
FIFO, 147
FIFO_RX, 147
getcFIFO_RX, 144
getcFIFO_TX, 145
initUART, 145
putcFIFO_RX, 146
putcFIFO_TX, 146
putsUART, 146
received_UART, 147
UART_RX_count, 148
UART_RX_count
main.c, 82
UART.h, 148
user.c, 149, 150
InitApp, 149
setLED, 149
user.h, 150, 154
BUFFER_FAIL, 151
BUFFER_SIZE, 152
BUFFER_SUCCESS, 152
InitApp, 154
LED0, 152
LED1, 152
LED2, 152
LED3, 152
SENSOR_TIME, 153
T0, 153
T1, 153
T2, 153
T3, 153

waitForBusyLCD
lcd_gpio.c, 65
lcd_gpio.h, 75
PMP.c, 108
PMP.h, 120
write
Buffer, 18
Buffer_I2C_FSM, 20
write_data_buffer_light
I2C.h, 58
main.c, 82
write_data_buffer_temp
I2C.h, 58
main.c, 82
writebuf
I2C_struct, 22
writeStrLCD
lcd_gpio.c, 66