

## Embedded 2 Dokumentation

Erzeugt von Doxygen 1.9.3



<b>1 Programmdokumentation</b>	<b>1</b>
1.0.1 Aufgabe I2C Lichtsensor	1
<b>2 Code-Style</b>	<b>3</b>
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
<b>3 Datenstruktur-Verzeichnis</b>	<b>5</b>
3.1 Datenstrukturen	5
<b>4 Datei-Verzeichnis</b>	<b>7</b>
4.1 Auflistung der Dateien	7
<b>5 Datenstruktur-Dokumentation</b>	<b>9</b>
5.1 Buffer Strukturreferenz	9
5.1.1 Ausführliche Beschreibung	9
5.1.2 Dokumentation der Felder	9
5.1.2.1 data	10
5.1.2.2 read	10
5.1.2.3 write	10
5.2 Buffer_I2C_FSM Strukturreferenz	10
5.2.1 Ausführliche Beschreibung	11
5.2.2 Dokumentation der Felder	11
5.2.2.1 data	11
5.2.2.2 read	11
5.2.2.3 write	11
5.3 I2C_struct Strukturreferenz	12
5.3.1 Ausführliche Beschreibung	12
5.3.2 Dokumentation der Felder	12
5.3.2.1 address	12
5.3.2.2 num_read	13
5.3.2.3 num_write	13
5.3.2.4 readbuf	13
5.3.2.5 status	13
5.3.2.6 writebuf	13
<b>6 Datei-Dokumentation</b>	<b>15</b>
6.1 configuration_bits.c-Dateireferenz	15
6.2 configuration_bits.c	15
6.3 I2C.c-Dateireferenz	16
6.3.1 Dokumentation der Funktionen	17

6.3.1.1	doI2C()	17
6.3.1.2	exchangeI2C()	18
6.3.1.3	FSM_Adresse_Read()	19
6.3.1.4	FSM_Adresse_Write()	20
6.3.1.5	FSM_Idle()	21
6.3.1.6	FSM_RECV_EN()	22
6.3.1.7	FSM_Repeated_Start()	23
6.3.1.8	FSM_Start()	24
6.3.1.9	FSM_Stop()	24
6.3.1.10	get_I2C_struct_FIFO()	25
6.3.1.11	initI2C()	26
6.3.1.12	print_sensor_values()	27
6.3.1.13	put_I2C_struct_FIFO()	27
6.4	I2C.c	28
6.5	I2C.h-Dateireferenz	32
6.5.1	Makro-Dokumentation	33
6.5.1.1	I2C_SCL	34
6.5.1.2	I2C_SCL_TRIS	34
6.5.1.3	I2C_SDA	34
6.5.1.4	I2C_SDA_TRIS	34
6.5.2	Dokumentation der benutzerdefinierten Typen	34
6.5.2.1	StateFunc	34
6.5.3	Dokumentation der Aufzählungstypen	34
6.5.3.1	i2c_status_t	34
6.5.4	Dokumentation der Funktionen	35
6.5.4.1	doI2C()	35
6.5.4.2	exchangeI2C()	36
6.5.4.3	FSM_Adresse_Read()	37
6.5.4.4	FSM_Adresse_Write()	38
6.5.4.5	FSM_Idle()	39
6.5.4.6	FSM_RECV_EN()	40
6.5.4.7	FSM_Repeated_Start()	41
6.5.4.8	FSM_Start()	42
6.5.4.9	FSM_Stop()	43
6.5.4.10	initI2C()	43
6.5.4.11	print_sensor_values()	44
6.5.5	Variablen-Dokumentation	45
6.5.5.1	FIFO_I2C	45
6.5.5.2	I2C_test_struct	45
6.5.5.3	read_data_buffer_light	45
6.5.5.4	read_data_buffer_temp	45
6.5.5.5	status_licht	45

6.5.5.6 status_temperatur . . . . .	45
6.5.5.7 trigger_FSM . . . . .	45
6.5.5.8 write_data_buffer_light . . . . .	46
6.5.5.9 write_data_buffer_temp . . . . .	46
6.6 I2C.h . . . . .	46
6.7 interrupts.c-Dateireferenz . . . . .	47
6.7.1 Dokumentation der Funktionen . . . . .	48
6.7.1.1 _T1Interrupt() . . . . .	48
6.8 interrupts.c . . . . .	48
6.9 main.c-Dateireferenz . . . . .	49
6.9.1 Makro-Dokumentation . . . . .	49
6.9.1.1 HEARTBEAT_MS . . . . .	49
6.9.1.2 MAIN . . . . .	50
6.9.2 Dokumentation der Funktionen . . . . .	50
6.9.2.1 main() . . . . .	50
6.10 main.c . . . . .	50
6.11 main_less.c-Dateireferenz . . . . .	51
6.11.1 Makro-Dokumentation . . . . .	52
6.11.1.1 BAUDRATE . . . . .	53
6.11.1.2 BRGVAL . . . . .	53
6.11.1.3 BUFFER_FAIL . . . . .	53
6.11.1.4 BUFFER_SIZE . . . . .	53
6.11.1.5 BUFFER_SUCCESS . . . . .	53
6.11.1.6 HEARTBEAT_MS . . . . .	53
6.11.1.7 I2C_SCL . . . . .	54
6.11.1.8 I2C_SCL_TRIS . . . . .	54
6.11.1.9 I2C_SDA . . . . .	54
6.11.1.10 I2C_SDA_TRIS . . . . .	54
6.11.2 Dokumentation der benutzerdefinierten Typen . . . . .	54
6.11.2.1 StateFunc . . . . .	54
6.11.3 Dokumentation der Funktionen . . . . .	54
6.11.3.1 _T1Interrupt() . . . . .	55
6.11.3.2 _U1TXInterrupt() . . . . .	55
6.11.3.3 delay_ms() . . . . .	55
6.11.3.4 FSM2_ACK_Receive() . . . . .	56
6.11.3.5 FSM2_Adresse() . . . . .	57
6.11.3.6 FSM2_Data_Receive() . . . . .	57
6.11.3.7 FSM2_Idle() . . . . .	58
6.11.3.8 FSM2_Start() . . . . .	58
6.11.3.9 FSM2_Stop() . . . . .	59
6.11.3.10 getcFIFO_TX() . . . . .	60
6.11.3.11 init_ms_t4() . . . . .	60

6.11.3.12 init_timer1()	60
6.11.3.13 initI2C()	61
6.11.3.14 initUART()	61
6.11.3.15 main()	62
6.11.3.16 putcFIFO_TX()	63
6.11.3.17 putcUART()	63
6.11.3.18 putsUART()	63
6.11.3.19 Temp_FSM2()	64
6.11.4 Variablen-Dokumentation	64
6.11.4.1 data	64
6.11.4.2 DELAY_ANPASSUNG	65
6.11.4.3 FIFO	65
6.12 main_less.c	65
6.13 system.c-Dateireferenz	70
6.13.1 Dokumentation der Funktionen	70
6.13.1.1 ConfigureOscillator()	71
6.13.1.2 delay_ms()	71
6.13.1.3 init_ms_t4()	71
6.13.1.4 init_timer1()	72
6.14 system.c	72
6.15 system.h-Dateireferenz	74
6.15.1 Makro-Dokumentation	74
6.15.1.1 FCY	74
6.15.1.2 SYS_FREQ	75
6.15.2 Dokumentation der Funktionen	75
6.15.2.1 ConfigureOscillator()	75
6.15.2.2 delay_ms()	75
6.15.2.3 init_ms_t4()	76
6.15.2.4 init_timer1()	76
6.15.3 Variablen-Dokumentation	76
6.15.3.1 DELAY_ANPASSUNG	77
6.16 system.h	77
6.17 traps.c-Dateireferenz	77
6.17.1 Dokumentation der Funktionen	78
6.17.1.1 _AddressError()	78
6.17.1.2 _DefaultInterrupt()	78
6.17.1.3 _MathError()	78
6.17.1.4 _OscillatorFail()	78
6.17.1.5 _StackError()	79
6.18 traps.c	79
6.19 UART.c-Dateireferenz	81
6.19.1 Dokumentation der Funktionen	81

6.19.1.1 _U1TXInterrupt()	82
6.19.1.2 getcFIFO_TX()	82
6.19.1.3 initUART()	82
6.19.1.4 putcFIFO_TX()	83
6.19.1.5 putcUART()	83
6.19.1.6 putsUART()	84
6.20 UART.c	84
6.21 UART.h-Dateireferenz	86
6.21.1 Makro-Dokumentation	87
6.21.1.1 BAUDRATE	87
6.21.1.2 BRGVAL	87
6.21.2 Dokumentation der Funktionen	87
6.21.2.1 getcFIFO_TX()	87
6.21.2.2 initUART()	88
6.21.2.3 putcFIFO_TX()	88
6.21.2.4 putsUART()	89
6.21.3 Variablen-Dokumentation	89
6.21.3.1 FIFO	89
6.22 UART.h	89
6.23 user.c-Dateireferenz	90
6.23.1 Dokumentation der Funktionen	91
6.23.1.1 InitApp()	91
6.23.1.2 setLED()	91
6.24 user.c	91
6.25 user.h-Dateireferenz	92
6.25.1 Makro-Dokumentation	93
6.25.1.1 BUFFER_FAIL	93
6.25.1.2 BUFFER_SIZE	93
6.25.1.3 BUFFER_SUCCESS	93
6.25.1.4 LED0	93
6.25.1.5 LED1	93
6.25.1.6 LED2	94
6.25.1.7 LED3	94
6.25.1.8 SENSOR_TIME	94
6.25.1.9 T0	94
6.25.1.10 T1	94
6.25.1.11 T2	94
6.25.1.12 T3	95
6.25.2 Dokumentation der Funktionen	95
6.25.2.1 InitApp()	95
6.26 user.h	95
6.27 Code Style.markdown-Dateireferenz	95

6.28 Dokumentation.markdown-Dateireferenz . . . . .	95
<b>Index</b>	<b>97</b>



# Kapitel 1

## Programmdokumentation

Die Dokumentation erfolgt getrennt nach den jeweiligen Aufgaben

### 1.0.1 Aufgabe I2C Lichtsensor

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

Da der 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 `exchangeI2C()` 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 `exchangeI2C()` getätigt. 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 Tranceive-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 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 verzögernden Millisekunden
 * @return ...
 */
```

## Kapitel 3

# Datenstruktur-Verzeichnis

### 3.1 Datenstrukturen

Hier folgt die Aufzählung aller Datenstrukturen mit einer Kurzbeschreibung:

Buffer	9
Buffer_I2C_FSM	10
I2C_struct	
Datenstruktur für die Kapselung aller benötigten Variablen, welche für ein character basiertes FIFO benötigt werden	12



# Kapitel 4

## Datei-Verzeichnis

### 4.1 Auflistung der Dateien

Hier folgt die Aufzählung aller Dateien mit einer Kurzbeschreibung:

configuration_bits.c	15
I2C.c	16
I2C.h	32
interrupts.c	47
main.c	49
main_less.c	51
system.c	70
system.h	74
traps.c	77
UART.c	81
UART.h	86
user.c	90
user.h	92





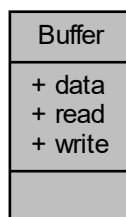
# Kapitel 5

## Datenstruktur-Dokumentation

### 5.1 Buffer Strukturreferenz

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

Zusammengehörigkeiten von Buffer:



#### Datenfelder

- uint8\_t [data](#) [[BUFFER\\_SIZE](#)]
- uint8\_t [read](#)
- uint8\_t [write](#)

#### 5.1.1 Ausführliche Beschreibung

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

#### 5.1.2 Dokumentation der Felder

### 5.1.2.1 data

```
uint8_t data
```

Definiert in Zeile 51 der Datei [main\\_less.c](#).

### 5.1.2.2 read

```
uint8_t read
```

Definiert in Zeile 52 der Datei [main\\_less.c](#).

### 5.1.2.3 write

```
uint8_t write
```

Definiert in Zeile 53 der Datei [main\\_less.c](#).

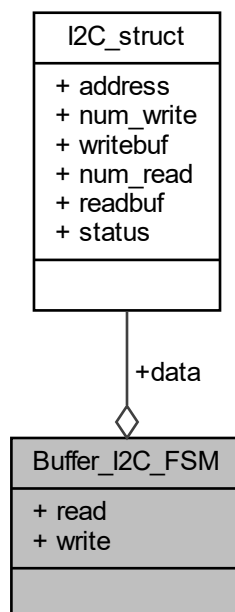
Die Dokumentation für diese Struktur wurde erzeugt aufgrund der Dateien:

- [main\\_less.c](#)
- [UART.h](#)

## 5.2 Buffer\_I2C\_FSM Strukturreferenz

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

Zusammengehörigkeiten von Buffer\_I2C\_FSM:



## Datenfelder

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

### 5.2.1 Ausführliche Beschreibung

Definiert in Zeile 52 der Datei `I2C.h`.

### 5.2.2 Dokumentation der Felder

#### 5.2.2.1 data

```
I2C_struct data[BUFFER_SIZE]
```

Definiert in Zeile 54 der Datei `I2C.h`.

#### 5.2.2.2 read

```
uint8_t read
```

Definiert in Zeile 55 der Datei `I2C.h`.

#### 5.2.2.3 write

```
uint8_t write
```

Definiert in Zeile 56 der Datei `I2C.h`.

Die Dokumentation für diese Struktur wurde erzeugt aufgrund der Datei:

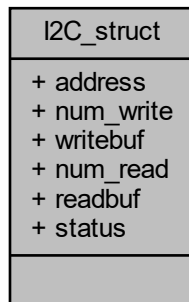
- `I2C.h`

## 5.3 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`

### 5.3.1 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 41 der Datei [I2C.h](#).

### 5.3.2 Dokumentation der Felder

#### 5.3.2.1 address

```
uint8_t address
```

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

#### 5.3.2.2 num\_read

```
uint16_t num_read
```

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

#### 5.3.2.3 num\_write

```
uint16_t num_write
```

Definiert in Zeile 44 der Datei [I2C.h](#).

#### 5.3.2.4 readbuf

```
uint8_t* readbuf
```

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

#### 5.3.2.5 status

```
i2c_status_t* status
```

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

#### 5.3.2.6 writebuf

```
uint8_t* writebuf
```

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

Die Dokumentation für diese Struktur wurde erzeugt aufgrund der Datei:

- [I2C.h](#)



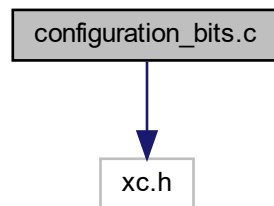
# Kapitel 6

## Datei-Dokumentation

### 6.1 configuration\_bits.c-Dateireferenz

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

Include-Abhängigkeitsdiagramm für configuration\_bits.c:



### 6.2 configuration\_bits.c

[gehe zur Dokumentation dieser Datei](#)

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

```

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

```

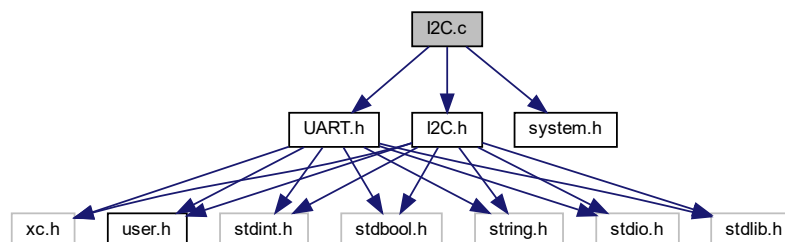
## 6.3 I2C.c-Dateireferenz

```

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

```

Include-Abhängigkeitsdiagramm für I2C.c:





## Funktionen

- `int16_t put_I2C_struct_FIFO (I2C_struct s)`  
*Legt eine I2C-Anfrage in dem I2C-FIFO ab.*
- `int16_t get_I2C_struct_FIFO (volatile I2C_struct *s)`  
*Entnimmt I2C-Anfrage aus dem I2C-FIFO.*
- `int16_t exchangeI2C (uint8_t address, uint16_t num_write, uint8_t *writebuf, uint16_t num_read, uint8_t *readbuf, i2c_status_t *status)`  
*Übergibt angeforderte I2C-Anfrage an das FIFO und liefert den aktuellen Status.*
- `void doI2C ()`  
*Wird jede ms in der Superloop ausgeführt und beinhalten die FSM für die I2C-Kommunikation.*
- `void initI2C ()`  
*Initialisiert die I2C-Kommunikation.*
- `void * FSM_Idle (void)`  
*Kopiert die Anfrage aus dem FIFO und leitet Start-Sequenz ein.*
- `void * FSM_Start (void)`  
*Beschreibt das Tranceive-Register mit der Adresse.*
- `void * FSM_Adresse_Write (void)`  
*Schreibt die zu übertragende Daten in das Tranceive-Register.*
- `void * FSM_Repeated_Start (void)`  
*Leitet einen Repeated Start ein und beschreibt das Tranceive Register mit der Adresse.*
- `void * FSM_Adresse_Read (void)`  
*Initiiert das Lesen der Daten des Slaves.*
- `void * FSM_RECV_EN (void)`  
*Auslesen des Receive Registers und Bestätigung mit ACK bzw.*
- `void * FSM_Stop (void)`  
*Überprüfung auf Abschluss der Stop-Sequenz und Rückkehr in den Idle-State.*
- `void print_sensor_values ()`  
*Ausgabe der ausgelesenen Sensor-Werte per UART.*

### 6.3.1 Dokumentation der Funktionen

#### 6.3.1.1 doI2C()

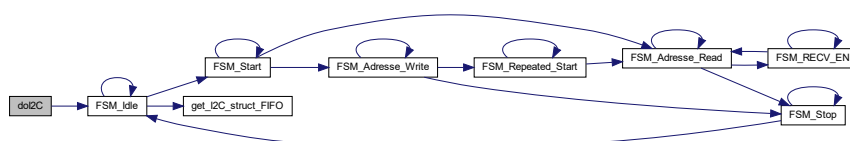
```
void doI2C (
    void )
```

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

Falls das FIFO neue Anfragen enthält wird die FSM getriggert.

Definiert in Zeile 103 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:



### 6.3.1.2 exchangeI2C()

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

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

#### Parameter

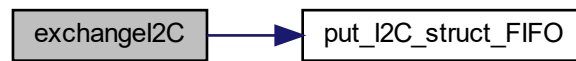
<i>address</i>	7 Bit Adresse des Slaves
<i>num_write</i>	Anzahl der zu sendenden Bytes, bei 0 keine Write Zugriff
<i>writebuf</i>	Zeiger auf zu schreibende Daten
<i>num_read</i>	Anzahl der zu lesenden Bytes, bei 0 keine Read Zugriff
<i>readbuf</i>	Zeiger auf Bereich, in welchem Daten abgespeichert werden sollen
<i>status</i>	Zeiger, um aktuellen Status zurückzugeben

#### Rückgabewerte

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

Definiert in Zeile 82 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:



### 6.3.1.3 FSM\_Adresse\_Read()

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

Initiiert das Lesen der Daten des Slaves.

#### Parameter

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

#### Rückgabe

FSM\_Stop, sobald ein Fehler bei der Kommunikation auftritt, z.B kein ACK vom Slave

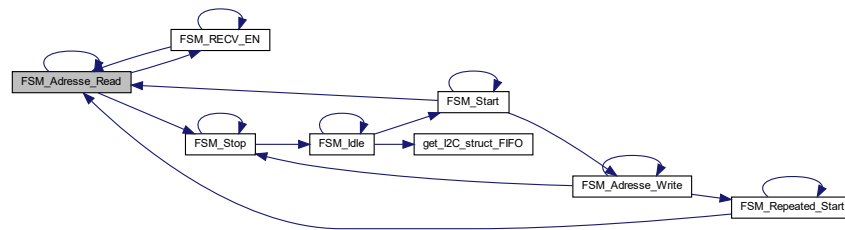
FSM\_RECV\_EN, sobald der Empfangsmodus für I2C aktiviert wurde

FSM\_Stop, sobald die Stop-Bedingungen an die Pins SDAx und SCLx weitergeleitet wurden.

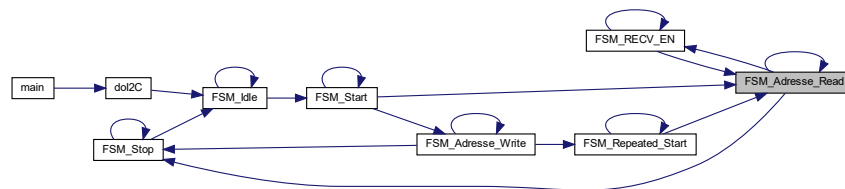
FSM\_Adresse\_Read, wenn kein ACK vom Slave erhalten oder das Bit der ACK-Sequenz nicht freigegeben ist

Definiert in Zeile [276](#) 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:



#### 6.3.1.4 FSM\_Adresse\_Write()

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

Schreibt die zu übertragende Daten in das Tranceive-Register.

##### Parameter

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

##### Rückgabe

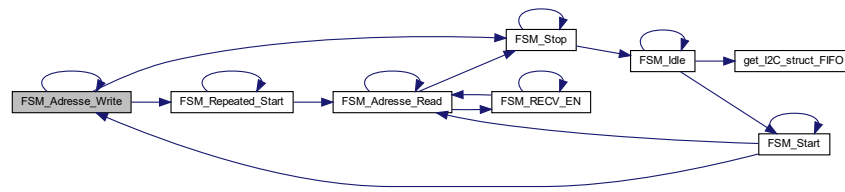
FSM\_Stop, sobald ein Fehler bei der Kommunikation auftritt, z.B kein ACK vom Slave

FSM\_Adresse\_Write, sobald keine Bytes mehr zu senden gibt

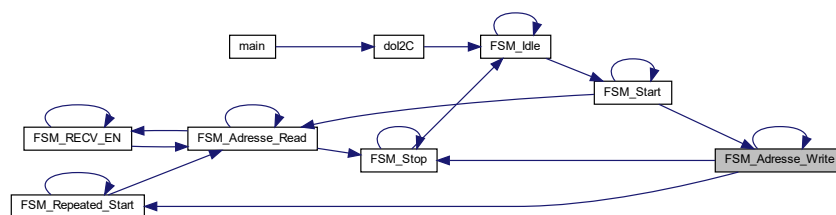
FSM\_Repeated\_Start, sobald die Bedingungen für den wiederholten Start an die Pins SDAx und SCLx weitergeleitet wurde.

Definiert in Zeile [211](#) 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:



### 6.3.1.5 FSM\_Idle()

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

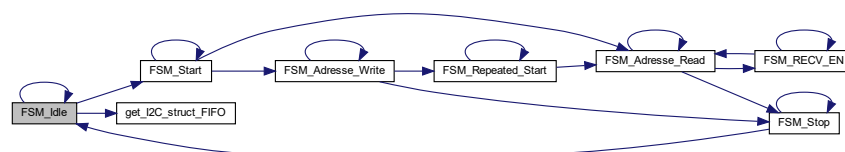
Kopiert die Anfrage aus dem FIFO und leitet Start-Sequenz ein.

#### Rückgabe

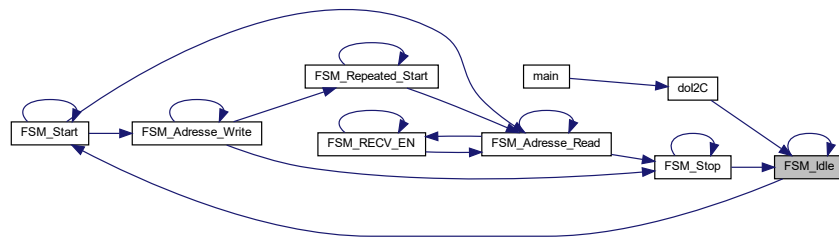
FSM\_Start, sobald die Startbedingungen an die Pins SDAx und SCLx weitergeleitet worden sind

Definiert in Zeile 159 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:



### 6.3.1.6 FSM\_RECEN()

```
void * FSM_RECEN (
    void )
```

Auslesen des Receive Registers und Bestätigung mit ACK bzw.

NACK

**Parameter**

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

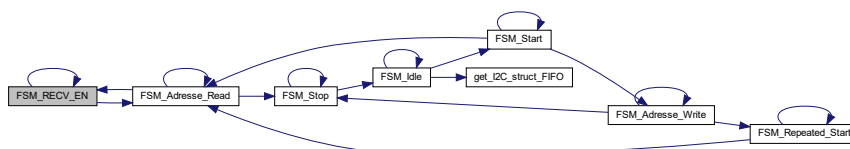
**Rückgabe**

FSM\_Adresse\_Read, sobald die Acknowledge-Sequenz an den Pins SDAx und SCLx initiiert wurde und das ACKDT Datenbit übertragen wurde

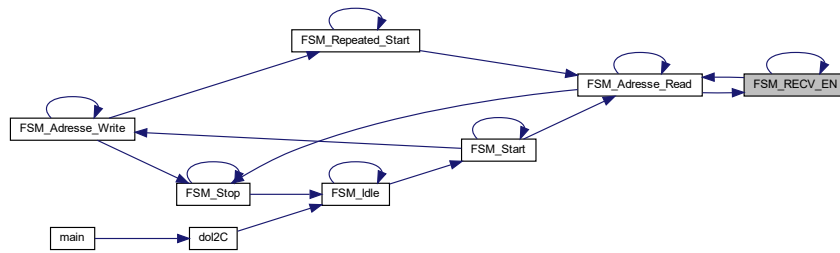
FSM\_RECEN, Wenn die Empfangssequenz nicht ausgeführt wurde

Definiert in Zeile [327](#) der Datei [I2C.c](#).

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



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



### 6.3.1.7 FSM\_Repeated\_Start()

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

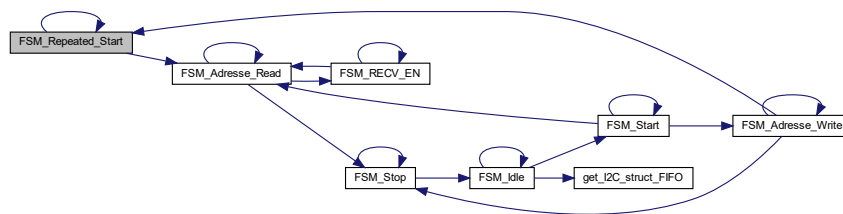
Leitet einen Repeated Start ein und beschreibt das Tranceive Register mit der Adresse.

Rückgabe

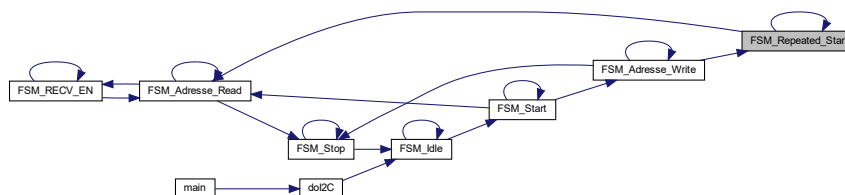
FSM\_Adresse\_Read, sobald es einen Restart gibt

Definiert in Zeile [253](#) 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:



### 6.3.1.8 FSM\_Start()

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

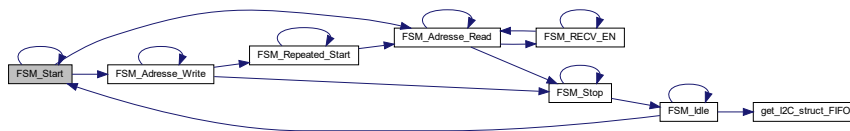
Beschreibt das Trancieve-Register mit der Adresse.

#### Rückgabe

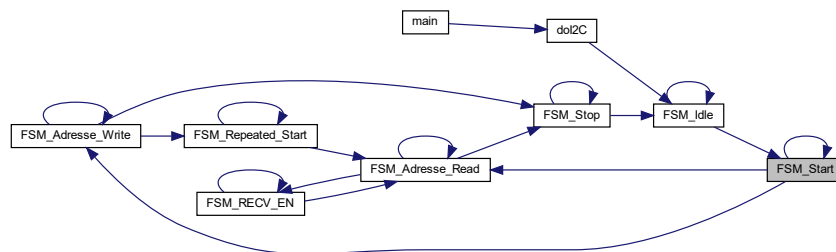
FSM\_Adresse\_Write, sobald geschrieben werden kann @retrun FSM\_Adresse\_Read, sobald gelesen werden kann

Definiert in Zeile 178 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:



### 6.3.1.9 FSM\_Stop()

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

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

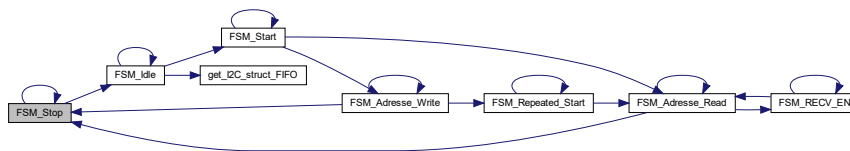


**Rückgabe**

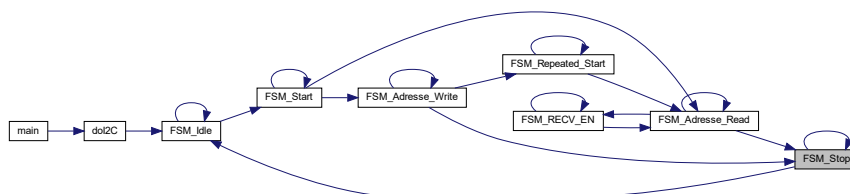
FSM\_Idle, wenn die Stop-Bedingungen erfolgreich an den Pins SDAx und SCLx weitergeleitet wurden  
 FSM\_Stop, wenn keine Stop-Bedingungen weitergeleitet wurden

Definiert in Zeile 359 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:

**6.3.1.10 get\_I2C\_struct\_FIFO()**

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

Entnimmt I2C-Anfrage aus dem I2C-FIFO.

**Parameter**

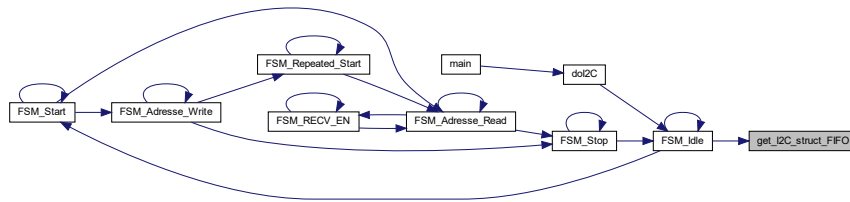
*s	Zeiger auf I2C-Anfrage in Form eines Structs des Typs <a href="#">I2C_struct</a>
----	--

**Rückgabewerte**

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

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

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



### 6.3.1.11 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 [112](#) 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:



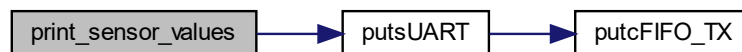
### 6.3.1.12 print\_sensor\_values()

```
void print_sensor_values (
    void )
```

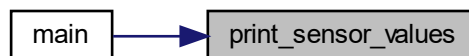
Ausgabe der ausgelesenen Sensor-Werte per UART.

Definiert in Zeile 372 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:



### 6.3.1.13 put\_I2C\_struct\_FIFO()

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

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

#### Parameter

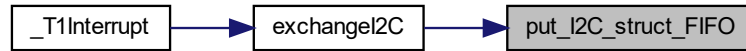
s	I2C-Anfrage in Form eines Structs des Typs <a href="#">I2C_struct</a>
---	---

#### Rückgabewerte

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

Definiert in Zeile 19 der Datei [I2C.c](#).

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



## 6.4 I2C.c

[gehe zur Dokumentation dieser Datei](#)

```

00001
00002 /* Files to Include
00003
00004 */
00005 #include "I2C.h"
00006 #include "system.h"
00007 #include "UART.h"
00008
00009
00010 /* Funktionen
00011 */
00012
00019 int16_t put_I2C_struct_FIFO(I2C_struct s)
00020 {
00021     _GIE=0;
00022     if ( ( FIFO_I2C.write + 1 == FIFO_I2C.read ) ||
00023         ( FIFO_I2C.read == 0 && FIFO_I2C.write + 1 == BUFFER_SIZE ) )
00024     {
00025         _GIE=1;
00026         return BUFFER_FAIL; // voll
00027     }
00028
00029     FIFO_I2C.data[FIFO_I2C.write] = s;
00030
00031     FIFO_I2C.write++;
00032     if (FIFO_I2C.write >= BUFFER_SIZE)
00033     {
00034         FIFO_I2C.write = 0;
00035     }
00036     _GIE=1;
00037     return BUFFER_SUCCESS;
00038
00039 } /* put_I2C_struct_FIFO() */
00040
00041
00048 int16_t get_I2C_struct_FIFO(volatile I2C_struct *s)
00049 {
00050     _GIE=0;
00051     if (FIFO_I2C.read == FIFO_I2C.write)
00052     {
00053         _GIE=1;
00054         return BUFFER_FAIL;
00055     }
00056     *s = FIFO_I2C.data[FIFO_I2C.read];
00057
00058     FIFO_I2C.read++;
00059     if (FIFO_I2C.read >= BUFFER_SIZE)
00060     {
00061         FIFO_I2C.read = 0;
00062     }
00063     _GIE=1;
00064     return BUFFER_SUCCESS;
00065
00066 } /* get_I2C_struct_FIFO() */
00067
00068
00082 int16_t exchangeI2C(uint8_t address, uint16_t num_write, uint8_t *writebuf, uint16_t num_read, uint8_t
    *readbuf, i2c_status_t *status)
00083 {
00084     I2C_struct temporary_struct = {address,num_write,writebuf,num_read,readbuf,status};
00085
  
```

```

00086     if (put_I2C_struct_FIFO(temporary_struct)) //Erfolgreich in FIFO abgelegt?
00087     {
00088         return 1;
00089     }
00090     else
00091     {
00092         return 0;
00093     }
00094
00095 } /* exchangeI2C() */
00096
00097
00103 void doI2C()
00104 {
00105     static StateFunc statefunc = FSM_Idle;
00106     statefunc = (StateFunc) (*statefunc) ();
00107 } /* doI2C() */
00108
00112 void initI2C()
00113 {
00114     I2C2CONbits.A10M = 0;
00115     I2C2BRG = 245; //100kHz
00116
00123     I2C_SDA_TRIS = 1;
00124     I2C_SCL_TRIS = 1;
00125     I2C_SDA = 0;
00126     I2C_SCL = 0;
00127
00128     int j;
00129     for (j=0; j<=9; j++) // takten bis min 1 Byte
00130     {
00131         I2C_SCL_TRIS = 0; delay_ms(1); // 5 us wären ausreichend ...100 kBaud
00132         I2C_SCL_TRIS = 1; delay_ms(1);
00133     }
00134     // Start Condition senden
00135     I2C_SCL_TRIS = 0; delay_ms(1);
00136     I2C_SDA_TRIS = 0; delay_ms(1);
00137     // Stop Condition senden
00138     I2C_SCL_TRIS = 1; delay_ms(1);
00139     I2C_SDA_TRIS = 1; delay_ms(1);
00140
00141     // Nun I2C erst anschalten
00142     _MI2C2IF = 0; //Interrupt falls noetig
00143     _MI2C2IE = 0;
00144
00149     I2C2CONbits.I2CEN = 1;
00150
00151 } /* initI2C() */
00152
00153
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     else
00167     {
00168         return FSM_Idle;
00169     }
00170 } /* *FSM_Idle() */
00171
00172
00178 void *FSM_Start(void)
00179 {
00180
00181     if (I2C2CONbits.SEN==0) // Wenn Startbedigungen erfüllt wurden
00182     {
00183         if (I2C_test_struct.num_write>0) //Schreiben
00184         {
00185             I2C2TRN=(I2C_test_struct.address<<1);
00186             return FSM_Adresse_Write;
00187         }
00188
00189         else if (I2C_test_struct.num_read>0) //Lesen
00190         {
00191             I2C2TRN=(I2C_test_struct.address<<1) | 0b1;
00192             return FSM_Adresse_Read;
00193         }
00194
00195     }
00196
00197     return FSM_Start;
00198
00199 } /* *FSM_Start() */
00200

```

```

00201
00211 void *FSM_Adresse_Write(void)
00212 {
00213     if (I2C2STATbits.TRSTAT==0) //Wenn erfolgreich übertragen
00214     {
00215         if (I2C2STATbits.ACKSTAT==1) // Leitet Stop-Bedingungen weiter
00216         {
00217             I2C2CONbits.PEN=1; //Fehler bei Kommunikation
00218             *I2C_test_struct.status=Error;
00219             return FSM_Stop;
00220         }
00221
00222         if (I2C2STATbits.ACKSTAT==0) //Wenn ACK von Slave erhalten
00223         {
00224             static int count=0;
00225
00226             if (count < I2C_test_struct.num_write) //Noch Bytes zu senden
00227             {
00228
00229                 I2C2TRN=I2C_test_struct.writebuf[count];
00230                 count++;
00231                 return FSM_Adresse_Write;
00232             }
00233
00234             else //Nichts mehr zu schicken
00235             {
00236                 count=0;
00237                 I2C2CONbits.RSEN=1; // Leitet Bedingungen für den Restart weiter
00238                 return FSM_Repeated_Start;
00239             }
00240
00241         }
00242     }
00243 }
00244 return FSM_Adresse_Write;
00245
00246 } /* *FSM_Adresse_Write() */
00247
00253 void *FSM_Repeated_Start(void)
00254 {
00255     if (I2C2CONbits.RSEN==0) // Wenn der Restart erfolgreich war
00256     {
00257         I2C2TRN=(I2C_test_struct.address<<1) | 0b1;
00258         return FSM_Adresse_Read;
00259     }
00260     return FSM_Repeated_Start;
00261 } /* *FSM_Repeated_Start() */
00262
00263
00264
00276 void *FSM_Adresse_Read(void)
00277 {
00278     if (I2C2STATbits.TRSTAT==0) //Wenn erfolgreich übertragen
00279     {
00280         if (I2C2STATbits.ACKSTAT==1) //Wenn NACK von Slave erhalten
00281         {
00282             I2C2CONbits.PEN=1; // Leitet Stop-Bedingungen weiter
00283             *I2C_test_struct.status=Error; //Fehler bei Kommunikation
00284             return FSM_Stop;
00285         }
00286
00287         if (I2C2STATbits.ACKSTAT==0) //Wenn ACK von Slave erhalten
00288         {
00289             if (I2C2CONbits.ACKEN==0) //Wenn Bit der ACK-Sequenz freigegeben
00290             {
00291                 static int count = 0;
00292
00293                 if (count < I2C_test_struct.num_read) //Noch Bytes zu empfangen
00294                 {
00295                     count++;
00296                     I2C2CONbits.RCEN=1; // Aktiviert den Empfangsmodus für I2C
00297                     return FSM_RECV_EN;
00298                 }
00299
00300                 else //Nichts mehr zu empfangen
00301                 {
00302                     count = 0;
00303                     I2C2CONbits.PEN=1; // Leitet Stop-Bedingungen weiter
00304                     *I2C_test_struct.status=Finished; //Anforderung abgearbeitet
00305                     return FSM_Stop;
00306                 }
00307             }
00308             else
00309             {
00310                 return FSM_Adresse_Read;
00311             }
00312

```

```

00313     }
00314 }
00315 }
00316 return FSM_Adresse_Read;
00317
00318 } /* *FSM_Adresse_Read() */
00319
00327 void *FSM_RECV_EN(void)
00328 {
00329     if (I2C2CONbits.RCEN==0)    //Wenn der Empfangsmodus aktiviert wurde
00330     {
00331         static int count = 0;
00332         I2C_test_struct.readbuf[count]=I2C2RCV;
00333         count++;
00334
00335         if (count>=I2C_test_struct.num_read) //Wenn letztes Byte empfangen wurde
00336         {
00337             count=0;
00338             I2C2CONbits.ACKDT=1;    //Sendet einen NACK während eines Acknowledge
00339         }
00340         else
00341         {
00342             I2C2CONbits.ACKDT=0;    //Sendet einen ACK während eines Acknowledge
00343         }
00344
00345         I2C2CONbits.ACKEN=1;    //Initiiert die Acknowledge-Sequenz
00346         return FSM_Adresse_Read;
00347     }
00348
00349     return FSM_RECV_EN;
00350
00351 } /* *FSM_RECV_EN() */
00352
00359 void *FSM_Stop(void)
00360 {
00361     if (I2C2CONbits.PEN==0)    //Wenn die Stop-Bedingungen weitergeleitet wurden
00362     {
00363         return FSM_Idle;
00364     }
00365     return FSM_Stop;
00366
00367 } /* *FSM_Stop() */
00368
00372 void print_sensor_values()
00373 {
00374     //Temperatur
00375     if (status_temperatur==Finished)
00376     {
00377         double temp = read_data_buffer_temp[0]<<8|read_data_buffer_temp[1];
00378         char str[16];
00379         sprintf(str,"Temperatur: %.1f Grad",temp/256);
00380         putsUART(str);
00381
00382         char lf[2];
00383         sprintf(lf, "\n");
00384
00385         putsUART(lf);
00386         status_temperatur=Pending;
00387     }
00388
00389     if (status_licht==Finished)
00390     {
00391         //Licht
00392         double light = read_data_buffer_light[0]<<8 | read_data_buffer_light[1];
00393         char str[16];
00394         sprintf(str,"Licht: %.1f lux",light/1.2);
00395         putsUART(str);
00396
00397         char lf[2];
00398         sprintf(lf, "\n");
00399
00400         putsUART(lf);
00401         status_licht=Pending;
00402     }
00403
00404     if (status_licht==Error || status_temperatur==Error)
00405     {
00406         char str[16];
00407         sprintf(str, "Fehler beim Auslesen eines Sensors!");
00408         putsUART(str);
00409
00410         char lf[2];
00411         sprintf(lf, "\n");
00412
00413         putsUART(lf);
00414     }
00415 }

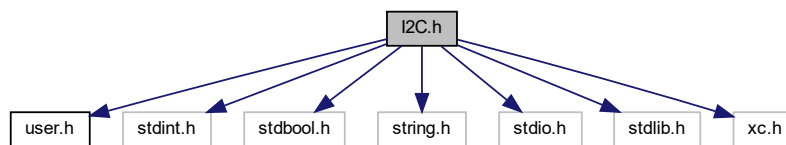
```

```
00416 } /* print_sensor_values() */
```

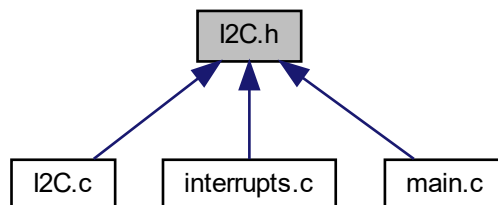
## 6.5 I2C.h-Dateireferenz

```
#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 [\\*\(\\* StateFunc\)](#) ()

## Aufzählungen

- enum [i2c\\_status\\_t](#) { [Pending](#) , [Finished](#) , [Error](#) }

## Funktionen

- int16\_t [exchangeI2C](#) (uint8\_t address, uint16\_t num\_write, uint8\_t \*writebuf, uint16\_t num\_read, uint8\_t \*readbuf, [i2c\\_status\\_t](#) \*status)  
*Übergibt angeforderte I2C-Anfrage an das FIFO und liefert den aktuellen Status.*
- void [doI2C](#) (void)  
*Wird jede ms in der Superloop ausgeführt und beinhalten die FSM für die I2C-Kommunikation.*
- void [initI2C](#) (void)  
*Initialisiert die I2C-Kommunikation.*
- void [print\\_sensor\\_values](#) (void)  
*Ausgabe der ausgelesenen Sensor-Werte per UART.*
- void \* [FSM\\_Idle](#) (void)  
*Kopiert die Anfrage aus dem FIFO und leitet Start-Sequenz ein.*
- void \* [FSM\\_Start](#) (void)  
*Beschreibt das Tranceive-Register mit der Adresse.*
- void \* [FSM\\_Adresse\\_Read](#) (void)  
*Initiiert das Lesen der Daten des Slaves.*
- void \* [FSM\\_Adresse\\_Write](#) (void)  
*Schreibt die zu übertragende Daten in das Tranceive-Register.*
- void \* [FSM\\_Repeated\\_Start](#) (void)  
*Leitet einen Repeated Start ein und beschreibt das Tranceive Register mit der Adresse.*
- void \* [FSM\\_RECV\\_EN](#) (void)  
*Auslesen des Receive Registers und Bestätigung mit ACK bzw.*
- void \* [FSM\\_Stop](#) (void)  
*Überprüfung auf Abschluss der Stop-Sequenz und Rückkehr in den Idle-State.*

## Variablen

- uint8\_t [write\\_data\\_buffer\\_temp](#)
- uint8\_t [write\\_data\\_buffer\\_light](#)
- uint8\_t [read\\_data\\_buffer\\_temp](#) [2]
- uint8\_t [read\\_data\\_buffer\\_light](#) [2]
- bool [trigger\\_FSM](#)
- [i2c\\_status\\_t](#) [status\\_temperatur](#)
- [i2c\\_status\\_t](#) [status\\_licht](#)
- [I2C\\_struct](#) [I2C\\_test\\_struct](#)
- [Buffer\\_I2C\\_FSM](#) [FIFO\\_I2C](#)

### 6.5.1 Makro-Dokumentation

#### 6.5.1.1 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 23 der Datei [I2C.h](#).

#### 6.5.1.2 I2C\_SCL\_TRIS

```
#define I2C_SCL_TRIS _TRISA2
```

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

#### 6.5.1.3 I2C\_SDA

```
#define I2C_SDA _RA3
```

Definiert in Zeile 24 der Datei [I2C.h](#).

#### 6.5.1.4 I2C\_SDA\_TRIS

```
#define I2C_SDA_TRIS _TRISA3
```

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

### 6.5.2 Dokumentation der benutzerdefinierten Typen

#### 6.5.2.1 StateFunc

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

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

### 6.5.3 Dokumentation der Aufzählungstypen

#### 6.5.3.1 i2c\_status\_t

```
enum i2c_status_t
```

## Aufzählungswerte

Pending	
Finished	
Error	

Definiert in Zeile 32 der Datei [I2C.h](#).

## 6.5.4 Dokumentation der Funktionen

### 6.5.4.1 doI2C()

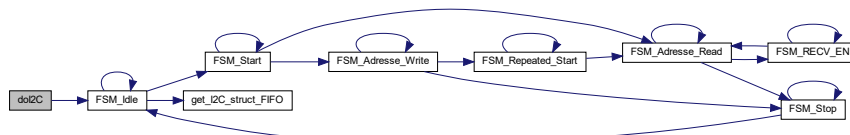
```
void doI2C (
    void )
```

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

Falls das FIFO neue Anfragen enthält wird die FSM getriggert.

Definiert in Zeile 103 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:



#### 6.5.4.2 exchangeI2C()

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

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

## Parameter

<i>address</i>	7 Bit Adresse des Slaves
<i>num_write</i>	Anzahl der zu sendenden 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

## Rückgabewerte

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

Definiert in Zeile [82](#) 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:



### 6.5.4.3 FSM\_Adresse\_Read()

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

Initiiert das Lesen der Daten des Slaves.

## Parameter

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

## Rückgabe

FSM\_Stop, sobald ein Fehler bei der Kommunikation auftritt, z.B kein ACK vom Slave

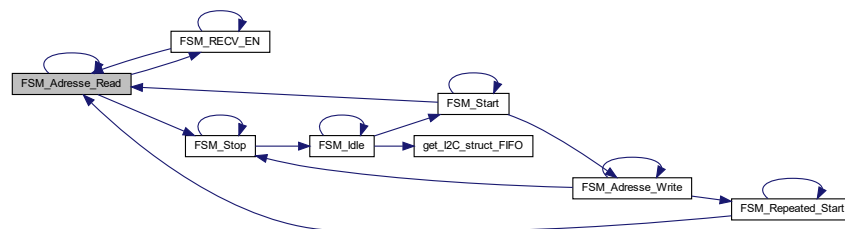
FSM\_RECV\_EN, sobald der Empfangsmodus für I2C aktiviert wurde

FSM\_Stop, sobald die Stop-Bedigungen an die Pins SDAx und SCLx weitergeleitet wurden.

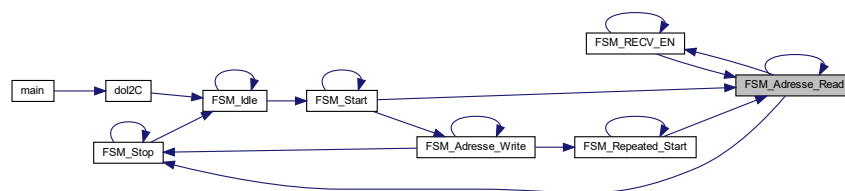
FSM\_Adresse\_Read, wenn kein ACK vom Slave erhalten oder das Bit der ACK-Sequenz nicht freigegeben ist

Definiert in Zeile [276](#) 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:



#### 6.5.4.4 FSM\_Adresse\_Write()

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

Schreibt die zu übertragende Daten in das Tranceive-Register.

## Parameter

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

## Rückgabe

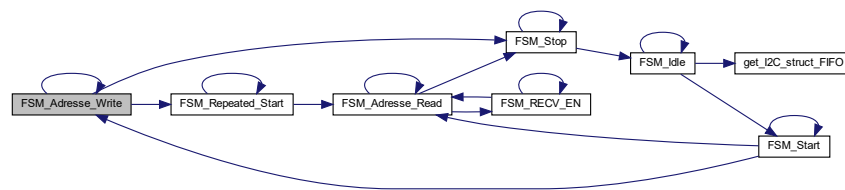
FSM\_Stop, sobald ein Fehler bei der Kommunikation auftritt, z.B kein ACK vom Slave

FSM\_Adresse\_Write, sobald keine Bytes mehr zu senden gibt

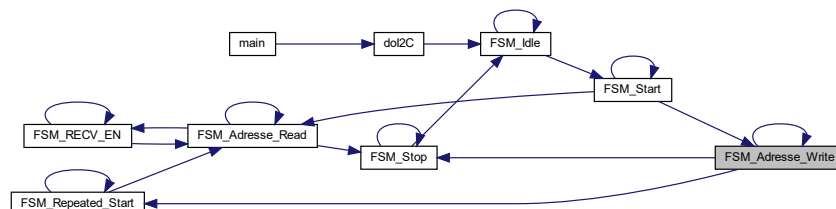
FSM\_Repeated\_Start, sobald die Bedingungen für den wiederholten Start an die Pins SDAx und SCLx weitergeleitet wurde.

Definiert in Zeile 211 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:



## 6.5.4.5 FSM\_Idle()

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

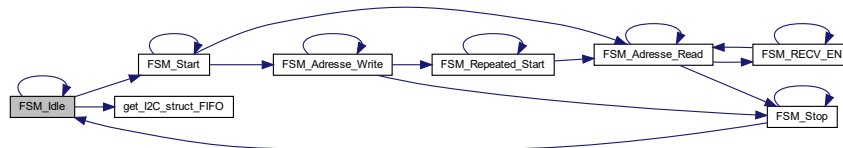
Kopiert die Anfrage aus dem FIFO und leitet Start-Sequenz ein.

**Rückgabe**

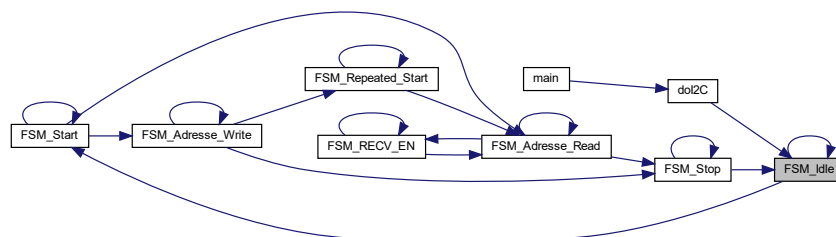
FSM\_Start, sobald die Startbedingungen an die Pins SDAx und SCLx weitergeleitet worden sind

Definiert in Zeile 159 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:

**6.5.4.6 FSM\_RECV\_EN()**

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

Auslesen des Receive Registers und Bestätigung mit ACK bzw.

NACK

**Parameter**

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

**Rückgabe**

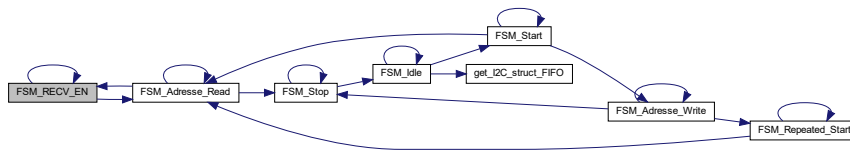
FSM\_Adresse\_Read, sobald die Acknowledge-Sequenz an den Pins SDAx und SCLx initiiert wurde und das ACKDT Datenbit übertragen wurde

FSM\_RECV\_EN, Wenn die Empfangssequenz nicht ausgeführt wurde

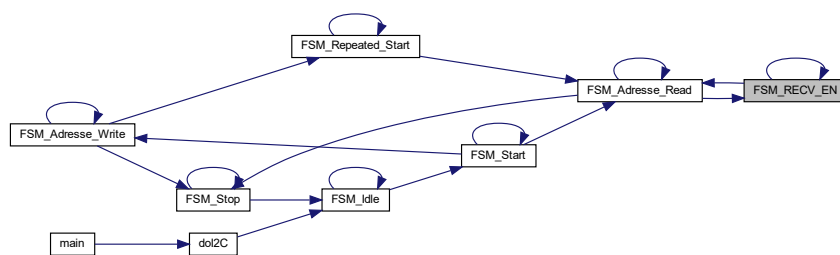


Definiert in Zeile 327 der Datei [I2C.c](#).

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



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



#### 6.5.4.7 FSM\_Repeated\_Start()

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

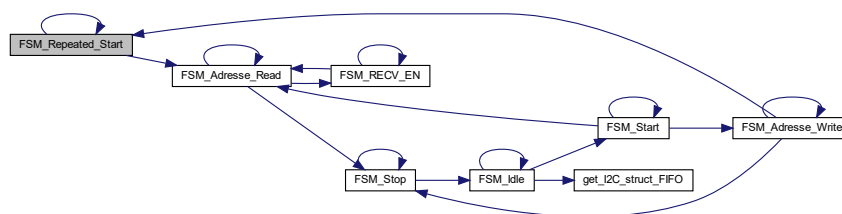
Leitet einen Repeated Start ein und beschreibt das Tranceive Register mit der Adresse.

##### Rückgabe

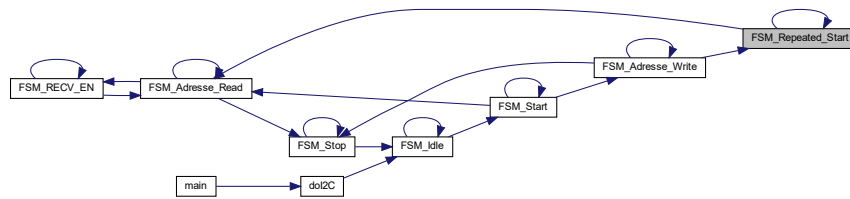
FSM\_Adresse\_Read, sobald es einen Restart gibt

Definiert in Zeile 253 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:



#### 6.5.4.8 FSM\_Start()

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

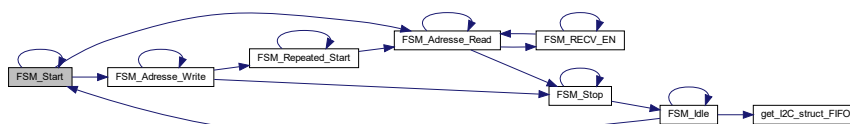
Beschreibt das Trancieve-Register mit der Adresse.

##### Rückgabe

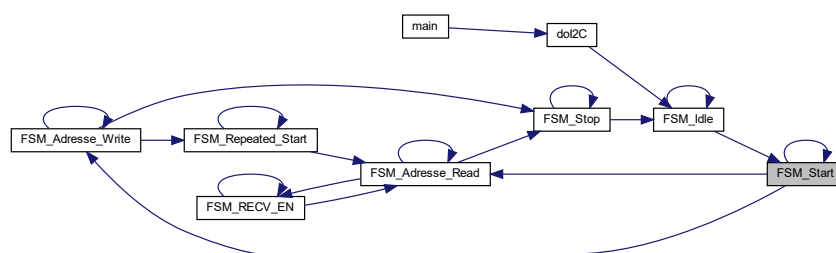
FSM\_Adresse\_Write, sobald geschrieben werden kann @retrun FSM\_Adresse\_Read, sobald gelesen werden kann

Definiert in Zeile 178 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:



#### 6.5.4.9 FSM\_Stop()

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

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

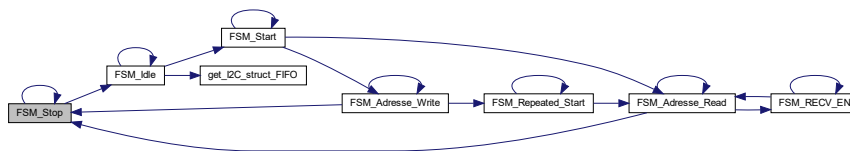
##### Rückgabe

FSM\_Idle, wenn die Stop-Bedingungen erfolgreich an den Pins SDAx und SCLx weitergeleitet wurden

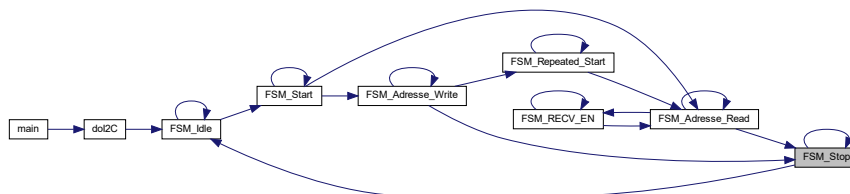
FSM\_Stop, wenn keine Stop-Bedingungen weitergeleitet wurden

Definiert in Zeile 359 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:



#### 6.5.4.10 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 112 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:



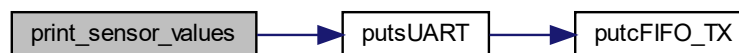
#### 6.5.4.11 `print_sensor_values()`

```
void print_sensor_values (  
    void )
```

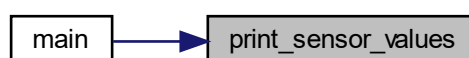
Ausgabe der ausgelesenen Sensor-Werte per UART.

Definiert in Zeile [372](#) 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:



## 6.5.5 Variablen-Dokumentation

### 6.5.5.1 FIFO\_I2C

```
Buffer_I2C_FSM FIFO_I2C [extern]
```

### 6.5.5.2 I2C\_test\_struct

```
I2C_struct I2C_test_struct [extern]
```

### 6.5.5.3 read\_data\_buffer\_light

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

### 6.5.5.4 read\_data\_buffer\_temp

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

### 6.5.5.5 status\_licht

```
i2c_status_t status_licht [extern]
```

### 6.5.5.6 status\_temperatur

```
i2c_status_t status_temperatur [extern]
```

### 6.5.5.7 trigger\_FSM

```
bool trigger_FSM [extern]
```

### 6.5.5.8 write\_data\_buffer\_light

```
uint8_t write_data_buffer_light [extern]
```

### 6.5.5.9 write\_data\_buffer\_temp

```
uint8_t write_data_buffer_temp [extern]
```

## 6.6 I2C.h

[gehe zur Dokumentation dieser Datei](#)

```
00001
00006
00007 /* Files to Include */
00008
00010 #include "user.h"
00011 // #include "UART.h"
00012 #include <stdint.h> /* Includes uint16_t definition */
00013 #include <stdbool.h> /* Includes true/false definition */
00014 #include <string.h>
00015 #include <stdio.h>
00016 #include <stdlib.h>
00017 #include <xc.h>
00018
00019
00020 /* Konstanten */
00021
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 /* Typedef */
00030
00032 typedef enum {Pending, Finished, Error} i2c_status_t;
00033
00041 typedef struct
00042 {
00043     uint8_t address;
00044     uint16_t num_write;
00045     uint8_t *writebuf;
00046     uint16_t num_read;
00047     uint8_t *readbuf;
00048     i2c_status_t *status;
00049 } I2C_struct;
00050
00051
00052 typedef struct
00053 {
00054     I2C_struct data[BUFFER_SIZE];
00055     uint8_t read; // zeigt auf das Feld mit dem ältesten Inhalt
00056     uint8_t write; // zeigt immer auf leeres Feld
00057 } Buffer_I2C_FSM;
00058
00059 typedef void (*StateFunc)();
00060
00061 #ifdef MAIN
00062
00063
00064 /* Global Variable Declaration */
00065
00067 uint8_t write_data_buffer_temp;
00068 uint8_t write_data_buffer_light;
00069 uint8_t read_data_buffer_temp[2];
00070 uint8_t read_data_buffer_light[2];
00071
00072 bool trigger_FSM;
00073
00074 i2c_status_t status_temperatur;
00075 i2c_status_t status_licht;
00076
```

```

00077
00078 I2C_struct I2C_test_struct = {0,0,NULL,0,NULL,NULL};
00079
00080 Buffer_I2C_FSM FIFO_I2C = {{},0,0}; //FIFO für die I2C FSM
00081 #else
00082 extern uint8_t write_data_buffer_temp;
00083 extern uint8_t write_data_buffer_light;
00084 extern uint8_t read_data_buffer_temp[2];
00085 extern uint8_t read_data_buffer_light[2];
00086
00087 extern bool trigger_FSM;
00088
00089 extern i2c_status_t status_temperatur;
00090 extern i2c_status_t status_licht;
00091
00092
00093 extern I2C_struct I2C_test_struct;
00094
00095 extern Buffer_I2C_FSM FIFO_I2C; //FIFO für die I2C FSM
00096 #endif
00097
00098
00099
00100
00101
00102 /* Prototypen */
00103
00105 int16_t exchangeI2C(uint8_t address, uint16_t num_write, uint8_t *writebuf, uint16_t num_read, uint8_t
    *readbuf, i2c_status_t *status);
00106
00107 void doI2C(void);
00108
00109 void initI2C(void);
00110
00111 void print_sensor_values(void);
00112
00113 void *FSM_Idle(void);
00114 void *FSM_Start(void);
00115 void *FSM_Adresse_Read(void);
00116 void *FSM_Adresse_Write(void);
00117 void *FSM_Repeated_Start(void);
00118 void *FSM_RECV_EN(void);
00119
00120 void *FSM_Stop(void);
00121
00122

```

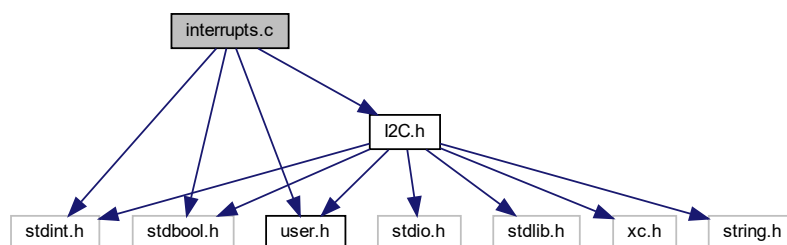
## 6.7 interrupts.c-Dateireferenz

```

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

```

Include-Abhängigkeitsdiagramm für interrupts.c:



## Funktionen

- void `_T1Interrupt` (void)

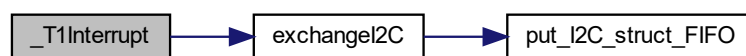
## 6.7.1 Dokumentation der Funktionen

### 6.7.1.1 \_T1Interrupt()

```
void _T1Interrupt (
    void )
```

Definiert in Zeile [25](#) der Datei [interrupts.c](#).

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



## 6.8 interrupts.c

[gehe zur Dokumentation dieser Datei](#)

```

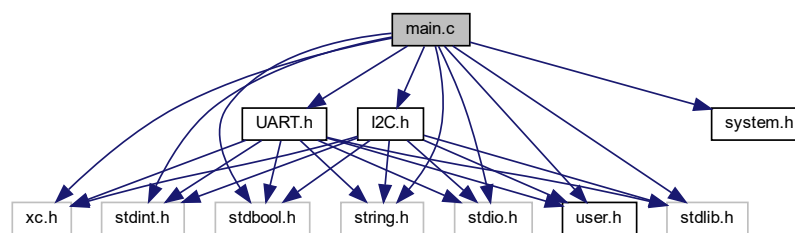
00001
00002 /* Files to Include */
00003
00004 /* Device header file */
00005 #if defined(__XC16__)
00006     #include <xc.h>
00007 #elif defined(__C30__)
00008     #if defined(__dsPIC33E__)
00009         #include <p33Exxxx.h>
00010     #elif defined(__dsPIC33F__)
00011         #include <p33Fxxx.h>
00012     #endif
00013 #endif
00014 #endif
00015
00016 #include <stdint.h> /* Includes uint16_t definition */
00017 #include <stdbool.h> /* Includes true/false definition */
00018 #include "user.h"
00019 #include "I2C.h"
00020
00021
00022 /* Interrupt Routines */
00023
00024 void __attribute__((__interrupt__, no_auto_psv)) _T1Interrupt(void)
00025 {
00026     _T1IF = 0; //Clear Timer1 interrupt flag
00027     static int count=0;
00028
00029     if (count>=SENSOR_TIME-1)
00030     {
00031         count=0;
00032         //Anfrage Temperatur-Sensor
00033         exchangeI2C(0b1001000, 1, &write_data_buffer_temp, 2, read_data_buffer_temp,
00034 &status_temperatur);
00035         //Anfrage Licht-Sensor
00036         exchangeI2C(0b0100011, 1, &write_data_buffer_light, 2, read_data_buffer_light, &status_licht);
00037     }
00038     else
00039     {
00040         count++;
00041     }
00042 }
00043 }
00044
00045
00046 /* TODO Add interrupt routine code here. */
```



## 6.9 main.c-Dateireferenz

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

Include-Abhängigkeitsdiagramm für main.c:



### Makrodefinitionen

- #define `MAIN`
- #define `HEARTBEAT_MS` 1

### Funktionen

- int16\_t `main` (void)

### 6.9.1 Makro-Dokumentation

#### 6.9.1.1 HEARTBEAT\_MS

```
#define HEARTBEAT_MS 1
```

Definiert in Zeile 30 der Datei `main.c`.

### 6.9.1.2 MAIN

```
#define MAIN
```

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

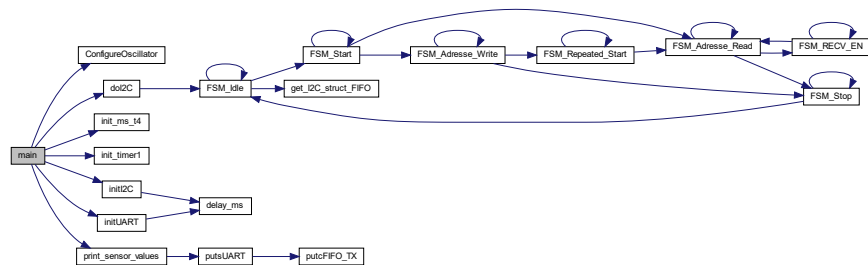
## 6.9.2 Dokumentation der Funktionen

### 6.9.2.1 main()

```
int16_t main (
    void )
```

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

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



## 6.10 main.c

[gehe zur Dokumentation dieser Datei](#)

```

00001 /*TODO
00002     Doku mit Doxygen
00003  *
00004  */
00005
00006
00007 /* Files to Include */
00008
00010 #define MAIN
00011 #include <xc.h>
00012
00013 #include <stdint.h> /* Includes uint16_t definition */
00014 #include <stdbool.h> /* Includes true/false definition */
00015 #include <string.h>
00016 #include <stdio.h>
00017 #include <stdlib.h>
00018
00019 #include "UART.h"
00020 #include "I2C.h"
00021
00022 #include "system.h" /* System funct/params, like osc/peripheral config */
00023 #include "user.h" /* User funct/params, such as InitApp */
00024
00025
00026
00027 /* Global Variable Declaration */
  
```

```

00028
00030 #define HEARTBEAT_MS 1
00031
00032
00033 /* Main Program */
00034
00036 int16_t main(void)
00037 {
00038     DELAY_ANPASSUNG = ((SYS_FREQ/96)*2180u11)/1000000u11; //Berechnung der Delay Anpassung
00039     uint16_t Count = 0;
00040
00041     ConfigureOscillator();
00042     initUART();
00043     init_timer1();
00044     init_ms_t4();
00045     initI2C();
00046
00047     _RP66R = _RPOUT_U1TX; //UART Pin Mapping
00048     RPINR18bits.U1RXR = 0b1011000;
00049
00050
00051     write_data_buffer_temp=0b00000000;
00052     write_data_buffer_light=0b00010000;
00053     while(1)
00054     {
00055         if(_T4IF)
00056         {
00057             _T4IF=0;
00058             Count++;
00059             if (Count >= HEARTBEAT_MS)
00060             {
00061                 Count = 0;
00062                 doI2C();
00063                 print_sensor_values();
00064             }
00065         }
00066     }
00067 }

```

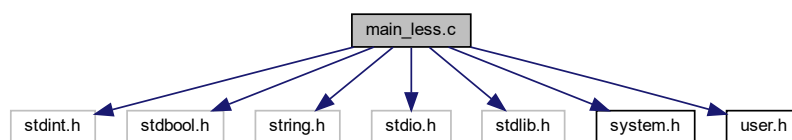
## 6.11 main\_less.c-Dateireferenz

```

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

```

Include-Abhängigkeitsdiagramm für main\_less.c:



## Datenstrukturen

- struct [Buffer](#)

## Makrodefinitionen

- `#define HEARTBEAT_MS 1`
- `#define BAUDRATE 9600`
- `#define BRGVAL ((FCY/BAUDRATE)/16)-1`
- `#define BUFFER_FAIL 0`
- `#define BUFFER_SUCCESS 1`
- `#define BUFFER_SIZE 128`
- `#define I2C_SCL_RA2`
- `#define I2C_SDA_RA3`
- `#define I2C_SCL_TRIS_TRISA2`
- `#define I2C_SDA_TRIS_TRISA3`

## Typdefinitionen

- `typedef void (* StateFunc) ()`

## Funktionen

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

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

- `void _T1Interrupt (void)`
- `void initUART ()`
- `void _U1TXInterrupt (void)`
- `int16_t putcUART (char c)`
- `void init_timer1 ()`
- `void initI2C ()`

*Initialisiert die I2C-Kommunikation.*

- `int16_t main (void)`

## Variablen

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

### 6.11.1 Makro-Dokumentation

### 6.11.1.1 BAUDRATE

```
#define BAUDRATE 9600
```

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

### 6.11.1.2 BRGVAL

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

Definiert in Zeile 33 der Datei [main\\_less.c](#).

### 6.11.1.3 BUFFER\_FAIL

```
#define BUFFER_FAIL 0
```

Definiert in Zeile 36 der Datei [main\\_less.c](#).

### 6.11.1.4 BUFFER\_SIZE

```
#define BUFFER_SIZE 128
```

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

### 6.11.1.5 BUFFER\_SUCCESS

```
#define BUFFER_SUCCESS 1
```

Definiert in Zeile 37 der Datei [main\\_less.c](#).

### 6.11.1.6 HEARTBEAT\_MS

```
#define HEARTBEAT_MS 1
```

Definiert in Zeile 29 der Datei [main\\_less.c](#).

#### 6.11.1.7 I2C\_SCL

```
#define I2C_SCL _RA2
```

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

#### 6.11.1.8 I2C\_SCL\_TRIS

```
#define I2C_SCL_TRIS _TRISA2
```

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

#### 6.11.1.9 I2C\_SDA

```
#define I2C_SDA _RA3
```

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

#### 6.11.1.10 I2C\_SDA\_TRIS

```
#define I2C_SDA_TRIS _TRISA3
```

Definiert in Zeile [45](#) der Datei [main\\_less.c](#).

### 6.11.2 Dokumentation der benutzerdefinierten Typen

#### 6.11.2.1 StateFunc

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

Definiert in Zeile [58](#) der Datei [main\\_less.c](#).

### 6.11.3 Dokumentation der Funktionen

### 6.11.3.1 \_T1Interrupt()

```
void _T1Interrupt (
    void )
```

Definiert in Zeile [91](#) der Datei [main\\_less.c](#).

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



### 6.11.3.2 \_U1TXInterrupt()

```
void _U1TXInterrupt (
    void )
```

Definiert in Zeile [136](#) der Datei [main\\_less.c](#).

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



### 6.11.3.3 delay\_ms()

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

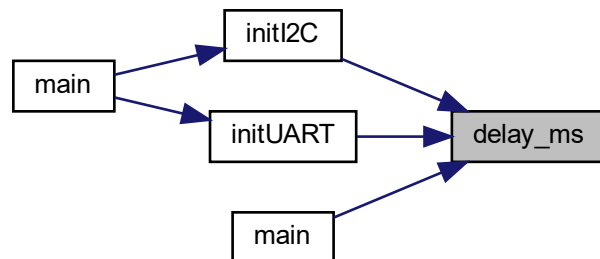
Delay in ms Blockierende Delay Funktion, eventuell nicht perfekte Verzögerung mit Hilfe einer for Schleife.

## Parameter

in	milliseconds	Verzögerungszeit in millisekunden
----	--------------	-----------------------------------

Definiert in Zeile 85 der Datei [main\\_less.c](#).

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

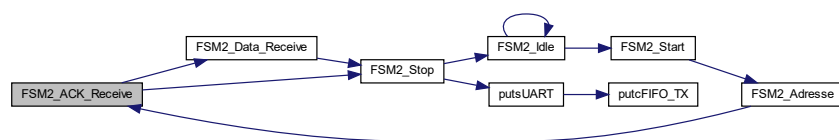


#### 6.11.3.4 FSM2\_ACK\_Receive()

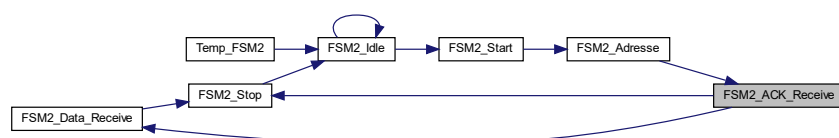
```
void * FSM2_ACK_Receive (
    void )
```

Definiert in Zeile 321 der Datei [main\\_less.c](#).

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



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



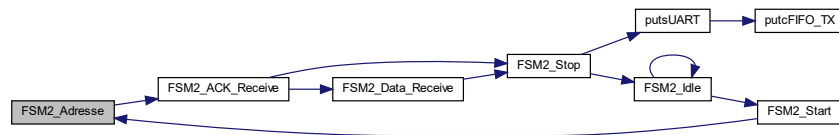


### 6.11.3.5 FSM2\_Adresse()

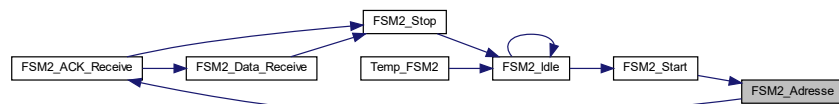
```
void * FSM2_Adresse (
    void )
```

Definiert in Zeile 313 der Datei [main\\_less.c](#).

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



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

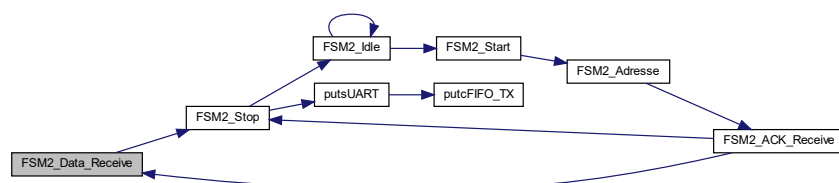


### 6.11.3.6 FSM2\_Data\_Receive()

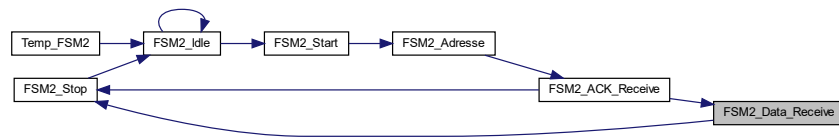
```
void * FSM2_Data_Receive (
    void )
```

Definiert in Zeile 330 der Datei [main\\_less.c](#).

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



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

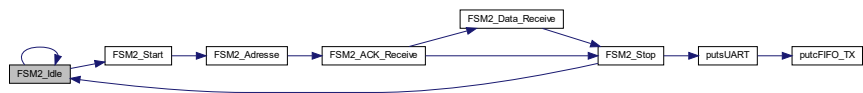


### 6.11.3.7 FSM2\_Idle()

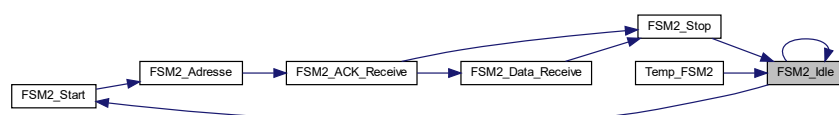
```
void * FSM2_Idle (
    void )
```

Definiert in Zeile [294](#) der Datei `main_less.c`.

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



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

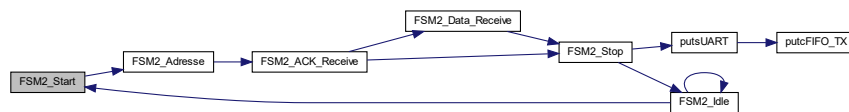


### 6.11.3.8 FSM2\_Start()

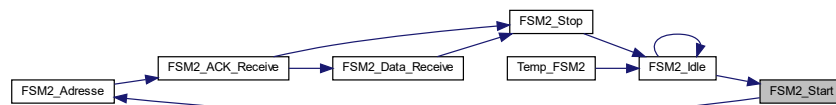
```
void * FSM2_Start (
    void )
```

Definiert in Zeile [306](#) der Datei `main_less.c`.

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



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

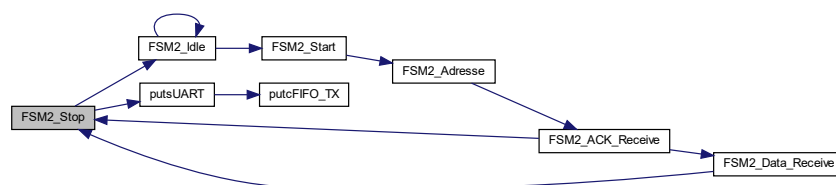


### 6.11.3.9 FSM2\_Stop()

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

Definiert in Zeile [352](#) der Datei [main\\_less.c](#).

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



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



#### 6.11.3.10 getcFIFO\_TX()

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

Definiert in Zeile 165 der Datei [main\\_less.c](#).

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



#### 6.11.3.11 init\_ms\_t4()

```
void init_ms_t4 (  
    void )
```

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

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

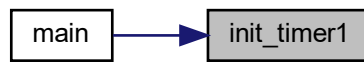


#### 6.11.3.12 init\_timer1()

```
void init_timer1 (  
    void )
```

Definiert in Zeile 210 der Datei [main\\_less.c](#).

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



### 6.11.3.13 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:



### 6.11.3.14 initUART()

```
void initUART (  
    void )
```

Definiert in Zeile [100](#) der Datei [main\\_less.c](#).

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



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

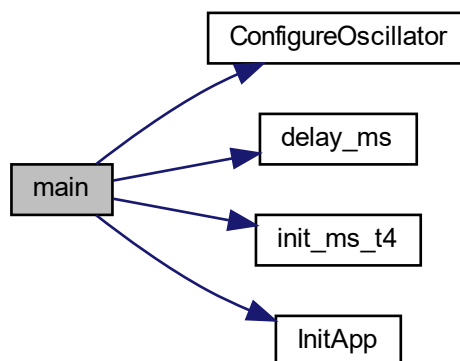


#### 6.11.3.15 main()

```
int16_t main (  
    void )
```

Definiert in Zeile [376](#) der Datei [main\\_less.c](#).

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

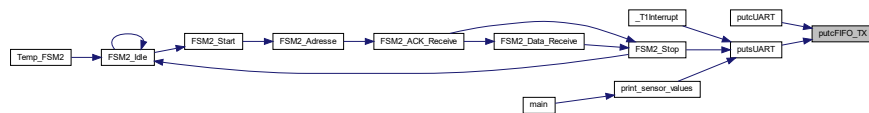


### 6.11.3.16 putcFIFO\_TX()

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

Definiert in Zeile 147 der Datei [main\\_less.c](#).

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



### 6.11.3.17 putcUART()

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

Definiert in Zeile 180 der Datei [main\\_less.c](#).

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



### 6.11.3.18 putsUART()

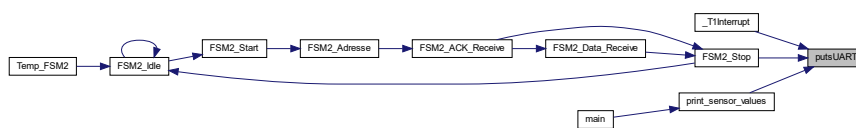
```
int16_t putsUART (
    const char * str )
```

Definiert in Zeile 190 der Datei [main\\_less.c](#).

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



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



### 6.11.3.19 Temp\_FSM2()

```
void Temp_FSM2 (
    void )
```

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

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



## 6.11.4 Variablen-Dokumentation

### 6.11.4.1 data

```
uint8_t data[2]
```

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



### 6.11.4.2 DELAY\_ANPASSUNG

uint32\_t DELAY\_ANPASSUNG

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

### 6.11.4.3 FIFO

Buffer FIFO = {{}, 0, 0}

Definiert in Zeile 56 der Datei [main\\_less.c](#).

## 6.12 main\_less.c

[gehe zur Dokumentation dieser Datei](#)

```

00001
00002 /* Files to Include */
00003
00005 /* Device header file */
00006 #if defined(__XC16__)
00007     #include <xc.h>
00008 #elif defined(__C30__)
00009     #if defined(__dsPIC33E__)
00010         #include <p33Exxxx.h>
00011     #elif defined(__dsPIC33F__)
00012         #include <p33Fxxx.h>
00013     #endif
00014 #endif
00015
00016
00017 #include <stdint.h> /* Includes uint16_t definition */
00018 #include <stdbool.h> /* Includes true/false definition */
00019 #include <string.h>
00020 #include <stdio.h>
00021 #include <stdlib.h>
00022
00023 #include "system.h" /* System funct/params, like osc/peripheral config */
00024 #include "user.h" /* User funct/params, such as InitApp */
00025
00026
00027 /* Global Variable Declaration */
00028
00029 #define HEARTBEAT_MS 1
00030 //UART
00031
00032 #define BAUDRATE 9600
00033 #define BRGVAL ((FCY/BAUDRATE)/16)-1
00034 //FIFO
00035
00036 #define BUFFER_FAIL 0
00037 #define BUFFER_SUCCESS 1
00038 #define BUFFER_SIZE 128
00039 uint32_t DELAY_ANPASSUNG;
00040
00041 //I2C
00042 #define I2C_SCL _RA2
00043 #define I2C_SDA _RA3
00044 #define I2C_SCL_TRIS _TRISA2
00045 #define I2C_SDA_TRIS _TRISA3
00046 uint8_t data[2];
00047
00048 /*Typen-Definitionen*****
00049
00050 typedef struct {
00051     uint8_t data[BUFFER_SIZE];
00052     uint8_t read; // zeigt auf das Feld mit dem ältesten Inhalt
00053     uint8_t write; // zeigt immer auf leeres Feld
00054 }Buffer;
00055
00056 Buffer FIFO = {{}, 0, 0};

```

```

00057
00058 typedef void *(*StateFunc)();
00059
00060
00061 /*Prototypes*****
00062 void init_ms_t4(void);
00063
00064 int16_t putsUART(const char *str);
00065 int16_t getcFIFO_TX(volatile uint16_t *c);
00066 //int16_t getcFIFO_RX(char *c);
00067
00068 int16_t putcFIFO_TX(char c);
00069 //int16_t putcFIFO_RX(char c);
00070
00071 void *FSM2_Idle(void);
00072 void *FSM2_Start(void);
00073 void *FSM2_Adresse(void);
00074 void *FSM2_ACK_Receive(void);
00075 void *FSM2_Data_Receive(void);
00076 void *FSM2_Stop(void);
00077 void Temp_FSM2(void);
00078
00079 /*Funktionen*****
00085 void delay_ms(uint16_t milliseconds) {
00086     uint32_t i=0;
00087     for (i=0;i<(DELAY_ANPASSUNG*(uint32_t)milliseconds);i++){
00088     }
00089 }
00090
00091 void __attribute__((__interrupt__, no_auto_psv)) _TlInterrupt(void)
00092 {
00093     _TlIF = 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.ALTI0 = 0;
00108
00109     U1MODEbits.URXINV = 0;
00110     U1MODEbits.RTSMD = 0;
00111
00112     U1MODEbits.BRGH = 0; // Standard-Speed mode
00113     U1BRG = BRGVAL; // Baud Rate setting for 9600
00114
00115     U1STAbits.UTXISEL0 = 0; // Interrupt after one TX character is transmitted
00116     U1STAbits.UTXISEL1 = 0;
00117     U1STAbits.UTXBRK = 0;
00118     U1STAbits.ADDEN = 0;
00119     U1STAbits.UTXINV = 0;
00120     U1STAbits.URXISEL = 0;
00121     U1STA = U1STA | 0b0001000000000000;
00122     //_URXEN = 1;
00123
00124     //_U1RXIE = 1; // Enable UART RX interrupt
00125
00126     U1MODEbits.UARTEN = 1; // Enable UART
00127     //delay_ms(2);
00128     U1STAbits.UTXEN = 1; // Enable UART TX
00129
00130     /* Wait at least 105 microseconds (1/9600) before sending first char */
00131     delay_ms(2);
00132     _U1TXIE = 1; // Enable UART TX interrupt
00133
00134 }
00135
00136 void __attribute__((__interrupt__)) _U1TXInterrupt(void)
00137 {
00138     _U1TXIF = 0; // Clear TX Interrupt flag
00139
00140     getcFIFO_TX(&U1TXREG);
00141
00142 }
00143
00144
00145
00146
00147 int16_t putcFIFO_TX(char c)
00148 {

```

```

00149 //if (buffer.write >= BUFFER_SIZE)
00150 //  buffer.write = 0; // erhöht sicherheit
00151 _LATF0 = 1;
00152 if ( ( FIFO.write + 1 == FIFO.read ) ||
00153      ( FIFO.read == 0 && FIFO.write + 1 == BUFFER_SIZE ) )
00154     return BUFFER_FAIL; // voll
00155
00156 FIFO.data[FIFO.write] = c;
00157
00158 FIFO.write++;
00159 if (FIFO.write >= BUFFER_SIZE)
00160     FIFO.write = 0;
00161
00162 return BUFFER_SUCCESS;
00163 }
00164
00165 int16_t getcFIFO_TX(volatile uint16_t *c)
00166 {
00167     _LATF0 = 1;
00168     if (FIFO.read == FIFO.write)
00169         return BUFFER_FAIL;
00170
00171     *c = FIFO.data[FIFO.read];
00172
00173     FIFO.read++;
00174     if (FIFO.read >= BUFFER_SIZE)
00175         FIFO.read = 0;
00176
00177     return BUFFER_SUCCESS;
00178 }
00179
00180 int16_t putcUART(char c){
00181     _LATF0 = 1;
00182     _GIE = 0; // Interrupts ausschalten
00183     int16_t erfolg = putcFIFO_TX(c);
00184     _GIE = 1;
00185     return erfolg;
00186
00187
00188 }
00189
00190 int16_t putsUART(const char *str) {
00191     _LATF0 = 1;
00192     uint16_t i;
00193     uint16_t length = strlen(str);
00194
00195     _GIE = 0; //Global Interrupt disable
00196     for(i = 0; i < length; i++) {
00197         //uint16_t ret = putcFIFO_TX(str[i]);
00198         if(! putcFIFO_TX(str[i]))
00199             break;
00200     }
00201     _GIE = 1;
00202     int16_t erfolg = -i;
00203     if(erfolg == -length)
00204         erfolg *= -1;
00205     _UITXIF = 1; //Interuppt Routine Starten um FIFO-Inhalt zu senden
00206     return erfolg;
00207 }
00208
00209 //Timer1
00210 void init_timer1(){
00211     __builtin_write_OSCCONL(0b00000011); //SOSC aktivieren
00212     TICONbits.TON = 0; // Disable Timer
00213     TICONbits.TCS = 1; // Select external clock
00214     TICONbits.TSYNC = 0; // Disable Synchronization
00215     TICONbits.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     TICONbits.TON = 1; // Start Timer
00223 }
00224
00225 //I2C
00226
00227 void Temp_FSM2(void)
00228 {
00229     static StateFunc statefunc = FSM2_Idle;
00230
00231     statefunc = (StateFunc) (*statefunc) ();
00232 }
00233
00234 void initI2C(){
00235     I2C2CONbits.A10M = 0;

```

```

00236     I2C2BRG = 245; //100kHz
00237
00238     // Einschalten I2C mit eigenem Workaround, I2C Peripheriemodul kann hier leider nicht verwendet
werden
00239     I2C_SDA_TRIS = 1;    // Pins wie einen Open-Kollektor-Treiber verwenden, d.h. 1 - hochohmig, 0
wird getrieben
00240     I2C_SCL_TRIS = 1;
00241     I2C_SDA = 0;
00242     I2C_SCL = 0;
00243
00244     int j;
00245     for (j=0; j<=9; j++)    // takten bis min 1 Byte
00246     {
00247         I2C_SCL_TRIS = 0; delay_ms(1); // 5 us wären ausreichend ...100 kBaud
00248         I2C_SCL_TRIS = 1; delay_ms(1);
00249     }
00250     // Start Condition senden
00251     I2C_SCL_TRIS = 0; delay_ms(1);
00252     I2C_SDA_TRIS = 0; delay_ms(1);
00253     // Stop Condition senden
00254     I2C_SCL_TRIS = 1; delay_ms(1);
00255     I2C_SDA_TRIS = 1; delay_ms(1);
00256
00257     // Nun I2C erst anschalten
00258     _MI2C2IF = 0; //Interrupt falls noetig
00259     _MI2C2IE = 0;
00260     I2C2CONbits.I2CEN = 1;
00261
00262     //Sensor Pointer auf TEMP Register setzten
00263     I2C2CONbits.SEN=1; //start
00264     while(I2C2CONbits.SEN==1){}
00265
00266     //Tx Device address + Write bit
00267     I2C2TRN=0b10010000;
00268     while(I2C2STATbits.TRSTAT==1){}
00269
00270     if (I2C2STATbits.ACKSTAT==1){    //if NACK received, generate stop condition and exit
00271         I2C2STATbits.ACKSTAT=0;
00272         I2C2CONbits.PEN=1;
00273         while(I2C2CONbits.PEN==1){} //wait for the stop interrupt;
00274         return;
00275     }
00276
00277     //Tx Register Address
00278     I2C2TRN=0b00000000; //Pointer auf TEMP REGISTER setzten
00279     while(I2C2STATbits.TRSTAT==1){}
00280
00281     if (I2C2STATbits.ACKSTAT==1){    //if NACK received, generate stop condition and exit
00282         I2C2STATbits.ACKSTAT=0;
00283         I2C2CONbits.PEN=1;
00284         while(I2C2CONbits.PEN==1){} //wait for the stop interrupt;
00285         return;
00286     }
00287
00288     I2C2CONbits.PEN=1; //stop
00289     while(I2C2CONbits.PEN==1){} //wait for the stop interrupt
00290 }
00291
00292
00293
00294 void *FSM2_Idle(void)
00295 {
00296     static int c = 0;
00297     if (c>=999){
00298         c=0;
00299         return FSM2_Start;
00300     }
00301     c++;
00302     return FSM2_Idle;
00303 }
00304
00305
00306 void *FSM2_Start(void)
00307 {
00308     I2C2CONbits.SEN=1; //Start
00309     while(I2C2CONbits.SEN==1){}
00310     return FSM2_Adresse;
00311 }
00312
00313 void *FSM2_Adresse(void)
00314 {
00315     //Tx Device address + Read bit
00316     I2C2TRN=0b10010001;
00317     while(I2C2STATbits.TRSTAT==1){} //Warten solange übertragen wird
00318     return FSM2_ACK_Receive;
00319 }
00320

```

```

00321 void *FSM2_ACK_Receive(void)
00322 {
00323     if (I2C2STATbits.ACKSTAT==1){ //if NACK received, generate stop condition and exit
00324         I2C2STATbits.ACKSTAT=0;
00325         return FSM2_Stop;
00326     }
00327     return FSM2_Data_Receive;
00328 }
00329
00330 void *FSM2_Data_Receive(void)
00331 {
00332     int N=2; //2 bytes empfangen
00333     int i;
00334
00335     for(i=0;i<N;i++){
00336         I2C2CONbits.RCEN=1; //Empfangen aktivieren
00337         while(I2C2CONbits.RCEN==1){} //RCEN cleared automatically when SSP1IF goes high
00338
00339         data[i]=I2C2RCV;
00340
00341         //ACK sequence
00342         if (i<N-1){ I2C2CONbits.ACKDT=0; } //jedes byte mit ACK bestätigen
00343         else {I2C2CONbits.ACKDT=1;} //send NACK if this is the last Byte
00344
00345         I2C2CONbits.ACKEN=1; //start ack/nack sequence
00346         while(I2C2CONbits.ACKEN==1){}
00347
00348     } //end for loop
00349     return FSM2_Stop;
00350 }
00351
00352 void *FSM2_Stop(void)
00353 {
00354     I2C2CONbits.PEN=1;
00355     while(I2C2CONbits.PEN==1){} //wait for the stop interrupt
00356
00357     float temp = data[0]<<8|data[1];
00358     char str[16];
00359     sprintf(str,"%f",temp/256);
00360     putsUART("Temperatur: ");
00361     putsUART(str);
00362     putsUART("°C");
00363     putsUART("\n");
00364
00365     return FSM2_Idle;
00366 }
00367
00368
00369
00370 /* Main Program */
00371
00372
00373 int16_t main(void)
00374 {
00375     DELAY_ANPASSUNG = ((SYS_FREQ/96)*2180u11)/1000000u11; //Berechnung der Delay Anpassung
00376     //uint16_t Count = 0;
00377     /* Configure the oscillator for the device */
00378     ConfigureOscillator();
00379     /* Initialize IO ports and peripherals */
00380     InitApp();
00381
00382     //initUART();
00383     //init_timer1();
00384     init_ms_t4();
00385     //initI2C();
00386
00387
00388     TRISBbits.TRISB8 = 0; //LED0 als Ausgang
00389     ANSELBbits.ANSB8 = 0; //LED0 als Digitaler Ausgang
00390
00391     TRISBbits.TRISB9 = 0; //LED als Ausgang
00392     ANSELBbits.ANSB9 = 0;
00393
00394     //Taster als Eingänge
00395     _TRISG12 = 1;
00396     //Pull-up Widerstände einschalten
00397     _CNPUG12 = 1;
00398
00399
00400
00401
00402
00403     _RP66R = _RPOUT_U1TX; //UART Pin Mapping
00404     RPINR18bits.U1RXR = 0b1011000;
00405     /* TODO <INSERT USER APPLICATION CODE HERE> */
00406
00407     while(1)
00408     {
00409         PORTBbits.RB8=1;
00410         delay_ms(200);

```

```

00411     PORTBbits.RB8=0;
00412     delay_ms(200);
00413     //if(_T4IF)
00414     //{
00415         //_T4IF=0;
00416         //Count++;
00417         //if (Count >= HEARTBEAT_MS)
00418         //{
00419             //Count = 0;
00420             //Temp_FSM2();
00421         //}
00422     //}
00423 }
00424 }
00425 }

```

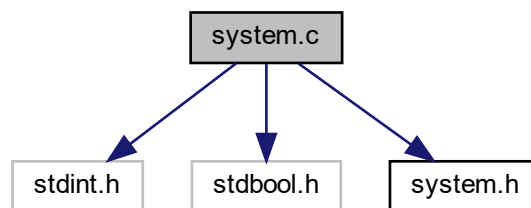
## 6.13 system.c-Dateireferenz

```

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

```

Include-Abhängigkeitsdiagramm für system.c:



### Funktionen

- void [ConfigureOscillator](#) (void)
- void [init\\_timer1](#) ()
- void [init\\_ms\\_t4](#) ()
- void [delay\\_ms](#) (uint16\_t milliseconds)

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

#### 6.13.1 Dokumentation der Funktionen

### 6.13.1.1 ConfigureOscillator()

```
void ConfigureOscillator (
    void )
```

Definiert in Zeile 40 der Datei [system.c](#).

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



### 6.13.1.2 delay\_ms()

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

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

Parameter

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

Definiert in Zeile 129 der Datei [system.c](#).

### 6.13.1.3 init\_ms\_t4()

```
void init_ms_t4 (
    void )
```

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

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



### 6.13.1.4 init\_timer1()

```
void init_timer1 (
    void )
```

Definiert in Zeile 98 der Datei [system.c](#).

## 6.14 system.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
00015
00016
00017 #include <stdint.h> /* For uint16_t definition */
00018 #include <stdbool.h> /* For true/false definition */
00019
00020 #include "system.h" /* variables/params used by system.c */
00021
00022
00023 /* System Level Functions */
00024 /*
00025  * Custom oscillator configuration funtions, reset source evaluation
00026  * functions, and other non-peripheral microcontroller initialization
00027  * functions get placed in system.c.
00028  */
00029
00030 /* Refer to the device Family Reference Manual Oscillator section for
00031  * information about available oscillator configurations. Typically
00032  * this would involve configuring the oscillator tuning register or clock
00033  * switching using the compiler's __builtin_write_OSCCON functions.
00034  * Refer to the C Compiler for PIC24 MCUs and dsPIC DSCs User Guide in the
00035  * compiler installation directory /doc folder for documentation on the
00036  * __builtin functions.*/
00037
00038 /* TODO Add clock switching code if appropriate. An example stub is below. */
00039 void ConfigureOscillator(void)
00040 {
00041     if (SYS_FREQ>7370000L) //Nur umschalten auf Primary (8 MHz) wenn höhere Frequenz erwünscht
00042     {
00043         switch (SYS_FREQ)
00044         {
00045             case 8000000L:
00046                 //PLL muss nicht konfiguriert werden
00047                 // externer Quartz mit 8Mhz
00048                 break;
00049             case 50000000L:
00050                 CLKDIVbits.PLLPOST=2; //N2=4
00051                 PLLFBD=48; //M=50
00052                 CLKDIVbits.PLLPRE=0; //N1=2
00053                 break;
00054             case 70000000L:
00055                 CLKDIVbits.PLLPOST=2; //N2=4
00056                 PLLFBD=188; //M=190
00057                 CLKDIVbits.PLLPRE=3; //N1=5
00058                 break;
00059             case 100000000L:
00060                 CLKDIVbits.PLLPOST=0; //N2=2
00061                 PLLFBD=123; //M=125
00062                 CLKDIVbits.PLLPRE=3; //N1=5
00063                 break;
00064             case 140000000L:
```



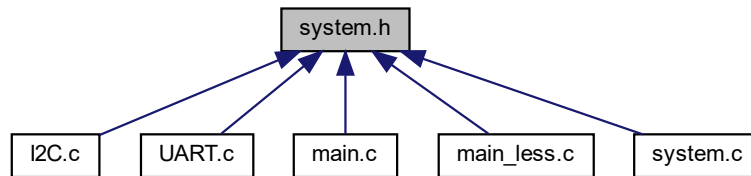
```

00066         CLKDIVbits.PLLPOST=0; //N2=2
00067         PLLFBD=173; //M=175
00068         CLKDIVbits.PLLPRE=3; //N1=5
00069         break;
00070     //default:
00071     //error Tets
00072 }
00073 OSCTUN = 0;
00074
00075 if (SYS_FREQ == 8000000L)
00076 {
00077     __builtin_write_OSCCONH(0x02); //Switch auf Primary ohne PLL
00078     __builtin_write_OSCCONL(OSCCON | 0x01);
00079     while (OSCCONbits.COSC!= 0x02); //Warten bis gewechselt wurde
00080 }
00081
00082 else
00083 {
00084     __builtin_write_OSCCONH(0x03); //Switch auf Primary mit PLL
00085     __builtin_write_OSCCONL(OSCCON | 0x01);
00086
00087     while (OSCCONbits.COSC!= 0x3); //Warten bis gewechselt wurde
00088     while (OSCCONbits.LOCK!= 1);
00089 }
00090
00091 }
00092
00093 }
00094 }
00095
00096
00097 //Timer1
00098 void init_timer1() //generiert in 1s Rythmus Interrupts
00099 {
00100     __builtin_write_OSCCONL(0b00000011); //SOSC aktivieren
00101     T1CONbits.TON = 0; // Disable Timer
00102     T1CONbits.TCS = 1; // Select external clock
00103     T1CONbits.TSYNC = 0; // Disable Synchronization
00104     T1CONbits.TCKPS = 0b00; // Select 1:1 Prescaler
00105     TMR1 = 0x00; // Clear timer register
00106     PR1 = 32767; // Load the period value, Quarztakt
00107
00108     IPC0bits.T1IP = 2; // Set Timer 1 Interrupt Priority Level
00109     IFS0bits.T1IF = 0; // Clear Timer 1 Interrupt Flag
00110     IEC0bits.T1IE = 1; // Enable Timer1 interrupt
00111     T1CONbits.TON = 1; // Start Timer
00112 }
00113
00114 void init_ms_t4() //Interrupt Flag wird jede ms gesetzt
00115 {
00116     T4CONbits.TON = 0; // Stop any 16/32-bit Timer3 operation
00117     T4CONbits.TCS = 0; // Select internal instruction cycle clock
00118
00119     T4CONbits.TGATE = 0; // Disable Gated Timer mode
00120     T4CONbits.TCKPS = 0b10; // Select 1:64 Prescaler
00121     TMR4 = 0x00; // Clear
00122     PR4 = (FCY/64000)-1; // Load 32-bit period value (lsw)
00123     //IFS0bits.T2IF = 0; // Clear Timer2 Interrupt Flag
00124     //IEC0bits.T2IE = 0; // Disable Timer2 interrupt
00125     T4CONbits.TON = 1; // Start 32-bit Timer
00126 }
00127
00128
00129 void delay_ms(uint16_t milliseconds)
00130 {
00131     uint32_t i=0;
00132     for (i=0;i<(DELAY_ANPASSUNG*(uint32_t)milliseconds);i++)
00133     {
00134     }
00135 }
00136

```

## 6.15 system.h-Dateireferenz

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



### Makrodefinitionen

- `#define SYS_FREQ 100000000L`
- `#define FCY SYS_FREQ/2`

### Funktionen

- void `ConfigureOscillator` (void)
- void `delay_ms` (uint16\_t milliseconds)  
*Delay in ms Blockierende Delay Funktion, eventuell nicht perfekte Verzögerung mit Hilfe einer for Schleife.*
- void `init_timer1` (void)
- void `init_ms_t4` (void)

### Variablen

- uint32\_t `DELAY_ANPASSUNG`

### 6.15.1 Makro-Dokumentation

#### 6.15.1.1 FCY

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

Definiert in Zeile 15 der Datei `system.h`.

### 6.15.1.2 SYS\_FREQ

```
#define SYS_FREQ 100000000L
```

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

## 6.15.2 Dokumentation der Funktionen

### 6.15.2.1 ConfigureOscillator()

```
void ConfigureOscillator (  
    void )
```

Definiert in Zeile 40 der Datei [system.c](#).

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



### 6.15.2.2 delay\_ms()

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

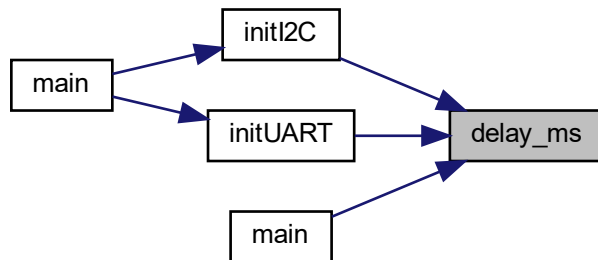
Delay in ms Blockierende Delay Funktion, eventuell nicht perfekte Verzögerung mit Hilfe einer for Schleife.

#### Parameter

in	<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:



#### 6.15.2.3 `init_ms_t4()`

```
void init_ms_t4 (  
    void )
```

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

#### 6.15.2.4 `init_timer1()`

```
void init_timer1 (  
    void )
```

Definiert in Zeile 210 der Datei [main\\_less.c](#).

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



### 6.15.3 Variablen-Dokumentation

### 6.15.3.1 DELAY\_ANPASSUNG

uint32\_t DELAY\_ANPASSUNG [extern]

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

## 6.16 system.h

[gehe zur Dokumentation dieser Datei](#)

```

00001
00002 /* System Level #define Macros */
00003
00004 /* TODO Define system operating frequency */
00005
00006
00007 /* Microcontroller MIPS (FCY) */
00008 // #define SYS_FREQ 7370000L
00009 // #define SYS_FREQ 8000000L
00010 // #define SYS_FREQ 50000000L
00011 // #define SYS_FREQ 70000000L
00012 #define SYS_FREQ 100000000L
00013 // #define SYS_FREQ 140000000L
00014
00015 #define FCY SYS_FREQ/2
00016
00017
00018 #ifdef MAIN
00019 uint32_t DELAY_ANPASSUNG;
00020 #else
00021 extern uint32_t DELAY_ANPASSUNG;
00022 #endif
00023
00024
00025
00026 /* System Function Prototypes */
00027
00028 /* Custom oscillator configuration funtions, reset source evaluation
00029 functions, and other non-peripheral microcontroller initialization functions
00030 go here. */
00031
00032
00033
00034 //System Prototypen
00035 void ConfigureOscillator(void); /* Handles clock switching/osc initialization */
00036 void delay_ms(uint16_t milliseconds);
00037
00038 void init_timer1(void);
00039 void init_ms_t4(void);
00040

```

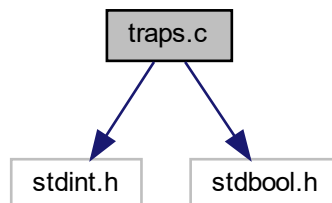
## 6.17 traps.c-Dateireferenz

```

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

```

Include-Abhängigkeitsdiagramm für traps.c:



## Funktionen

- void [\\_OscillatorFail](#) (void)
- void [\\_AddressError](#) (void)
- void [\\_StackError](#) (void)
- void [\\_MathError](#) (void)
- void [\\_DefaultInterrupt](#) (void)

### 6.17.1 Dokumentation der Funktionen

#### 6.17.1.1 [\\_AddressError\(\)](#)

```
void _AddressError (  
    void )
```

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

#### 6.17.1.2 [\\_DefaultInterrupt\(\)](#)

```
void _DefaultInterrupt (  
    void )
```

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

#### 6.17.1.3 [\\_MathError\(\)](#)

```
void _MathError (  
    void )
```

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

#### 6.17.1.4 [\\_OscillatorFail\(\)](#)

```
void _OscillatorFail (  
    void )
```

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

## 6.17.1.5 \_StackError()

```
void _StackError (
    void )
```

Definiert in Zeile 87 der Datei [traps.c](#).

## 6.18 traps.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 <p33Fxxx.h>
00012     #endif
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
00022 /* <Other function prototypes for debugging trap code may be inserted here> */
00023
00024 /* Use if INTCN2 ALTIPT=1 */
00025 void __attribute__((interrupt,no_auto_psv)) _OscillatorFail(void);
00026 void __attribute__((interrupt,no_auto_psv)) _AddressError(void);
00027 void __attribute__((interrupt,no_auto_psv)) _StackError(void);
00028 void __attribute__((interrupt,no_auto_psv)) _MathError(void);
00029
00030 #if defined(__HAS_DMA__)
00031 void __attribute__((interrupt,no_auto_psv)) _DMACError(void);
00032 #endif
00033
00034 #if defined(__dsPIC33F__)
00035 /* Use if INTCN2 ALTIPT=0 */
00036 void __attribute__((interrupt,no_auto_psv)) _AltOscillatorFail(void);
00037 void __attribute__((interrupt,no_auto_psv)) _AltAddressError(void);
00038 void __attribute__((interrupt,no_auto_psv)) _AltStackError(void);
00039 void __attribute__((interrupt,no_auto_psv)) _AltMathError(void);
00040
00041 #if defined(__HAS_DMA__)
00042 void __attribute__((interrupt,no_auto_psv)) _AltDMACError(void);
00043 #endif
00044 #endif
00045
00046 /* Default interrupt handler */
00047 void __attribute__((interrupt,no_auto_psv)) _DefaultInterrupt(void);
00048
00049 #if defined(__dsPIC33E__)
00050 /* These are additional traps in the 33E family. Refer to the PIC33E
00051 migration guide. There are no Alternate Vectors in the 33E family. */
00052 void __attribute__((interrupt,no_auto_psv)) _HardTrapError(void);
00053 void __attribute__((interrupt,no_auto_psv)) _SoftTrapError(void);
00054 #endif
00055
00056 /* Trap Handling */
00057
00058 /* These trap routines simply ensure that the device continuously loops
00059 within each routine. Users who actually experience one of these traps
00060 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 void __attribute__((interrupt,no_auto_psv)) _DMACError(void)
00101 {
00102     INTCON1bits.DMACERR = 0;          /* Clear the trap flag */
00103     while(1);
00104 }
00105 }
00106 #endif
00107
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 void __attribute__((interrupt,no_auto_psv)) _AltDMACError(void)
00138 {
00139     INTCON1bits.DMACERR = 0;          /* Clear the trap flag */
00140     while(1);
00141 }
00142 }
00143 #endif
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

```

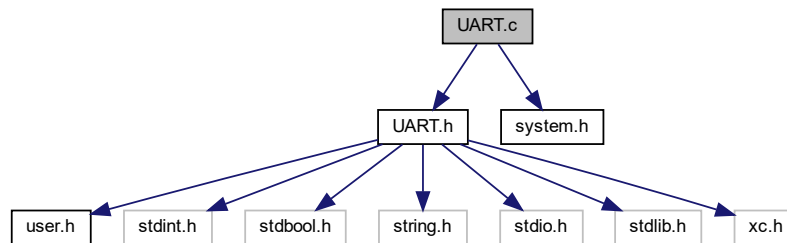
## 6.19 UART.c-Dateireferenz

```

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

```

Include-Abhängigkeitsdiagramm für UART.c:



### Funktionen

- void `initUART` ()
- void `_U1TXInterrupt` (void)
- int16\_t `putcFIFO_TX` (char c)
- int16\_t `getcFIFO_TX` (volatile uint16\_t \*c)
- int16\_t `putcUART` (char c)
- int16\_t `putsUART` (const char \*str)

#### 6.19.1 Dokumentation der Funktionen

### 6.19.1.1 \_U1TXInterrupt()

```
void _U1TXInterrupt (  
    void )
```

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

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



### 6.19.1.2 getcFIFO\_TX()

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

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

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



### 6.19.1.3 initUART()

```
void initUART (  
    void )
```

Definiert in Zeile 12 der Datei [UART.c](#).

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

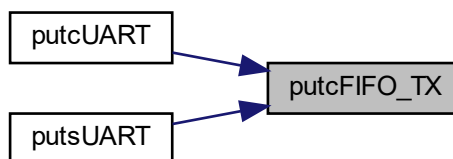


#### 6.19.1.4 putcFIFO\_TX()

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

Definiert in Zeile 57 der Datei [UART.c](#).

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



#### 6.19.1.5 putcUART()

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

Definiert in Zeile 102 der Datei [UART.c](#).

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



### 6.19.1.6 putsUART()

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

Definiert in Zeile 112 der Datei [UART.c](#).

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



## 6.20 UART.c

[gehe zur Dokumentation dieser Datei](#)

```

00001
00002 /* Files to Include                                */
00003
00005 #include "UART.h"
00006 #include "system.h"
00007
00008
00009 /* Funktionen                                        */
00010
00012 void initUART()
00013 {
00014     U1MODEbits.STSEL = 0; // 1-Stop bit
00015     U1MODEbits.PDSEL = 0; // No Parity, 8-Data bits
00016     U1MODEbits.ABAUD = 0; // Auto-Baud disabled
00017     U1MODEbits.UEN = 0;
00018     U1MODEbits.LPBACK = 0;
00019     U1MODEbits.RXINV = 0;
00020     //U1MODEbits.ALTI0 = 0;
00021
00022     U1MODEbits.URXINV = 0;
00023     U1MODEbits.RTSMD = 0;
00024
00025     U1MODEbits.BRGH = 0; // Standard-Speed mode
00026     U1BRG = BRGVAL; // Baud Rate setting for 9600
00027
00028     U1STAbits.UTXISEL0 = 0; // Interrupt after one TX character is transmitted
00029     U1STAbits.UTXISEL1 = 0;
00030     U1STAbits.UTXBRK = 0;
00031     U1STAbits.ADDEN = 0;
00032     U1STAbits.UTXINV = 0;
00033     U1STAbits.URXISEL = 0;
00034     U1STA = U1STA | 0b0001000000000000;
00035     //_URXEN = 1;
00036
00037     //_U1RXIE = 1; // Enable UART RX interrupt
00038
00039     U1MODEbits.UARTEN = 1; // Enable UART
00040     delay_ms(2);
00041     U1STAbits.UTXEN = 1; // Enable UART TX
00042
00043     /* Wait at least 105 microseconds (1/9600) before sending first char */
00044     delay_ms(2);
00045     _U1TXIE = 1; // Enable UART TX interrupt
00046
00047 } /* initUART() */
00048
00049
00050 void __attribute__((__interrupt__, no_auto_psv)) _U1TXInterrupt(void)
00051 {
```

```

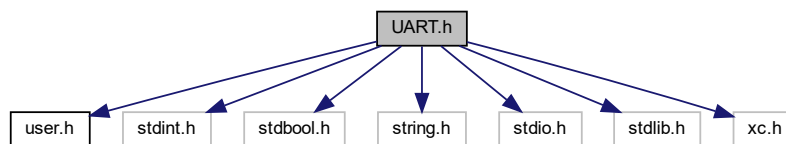
00052     _UITXIF = 0; // Clear TX Interrupt flag
00053     getcFIFO_TX(&UITXREG);
00054
00055 }
00056
00057 int16_t putcFIFO_TX(char c)
00058 {
00059     _GIE=0;
00060     //if (buffer.write >= BUFFER_SIZE)
00061     // buffer.write = 0; // erhöht sicherheit
00062     _LATF0 = 1;
00063     if ( ( FIFO.write + 1 == FIFO.read ) ||
00064         ( FIFO.read == 0 && FIFO.write + 1 == BUFFER_SIZE ) )
00065     {
00066         _GIE=1;
00067         return BUFFER_FAIL; // voll
00068     }
00069
00070     FIFO.data[FIFO.write] = c;
00071
00072     FIFO.write++;
00073     if (FIFO.write >= BUFFER_SIZE)
00074     {
00075         FIFO.write = 0;
00076     }
00077     _GIE=1;
00078     return BUFFER_SUCCESS;
00079
00080 } /* putcFIFO_TX() */
00081
00082 int16_t getcFIFO_TX(volatile uint16_t *c)
00083 {
00084     _GIE=0;
00085     if (FIFO.read == FIFO.write)
00086     {
00087         _GIE=1;
00088         return BUFFER_FAIL;
00089     }
00090     *c = FIFO.data[FIFO.read];
00091
00092     FIFO.read++;
00093     if (FIFO.read >= BUFFER_SIZE)
00094     {
00095         FIFO.read = 0;
00096     }
00097     _GIE=1;
00098     return BUFFER_SUCCESS;
00099
00100 } /* getcFIFO_TX() */
00101
00102 int16_t putcUART(char c)
00103 {
00104     _LATF0 = 1;
00105     _GIE = 0; // Interrupts ausschalten
00106     int16_t erfolg = putcFIFO_TX(c);
00107     _GIE = 1;
00108     return erfolg;
00109
00110 } /* putcUART() */
00111
00112 int16_t putsUART(const char *str)
00113 {
00114     uint16_t i;
00115     uint16_t length = strlen(str);
00116
00117     _GIE = 0; //Global Interrupt disable
00118     for(i = 0; i < length; i++)
00119     {
00120         if(! putcFIFO_TX(str[i]))
00121             break;
00122     }
00123     _GIE = 1;
00124     int16_t erfolg = -i;
00125     if(erfolg == -length)
00126         erfolg *= -1;
00127     _UITXIF = 1; //Interuppt Routine Starten um FIFO-Inhalt zu senden
00128     return erfolg;
00129
00130 } /* putsUART() */

```

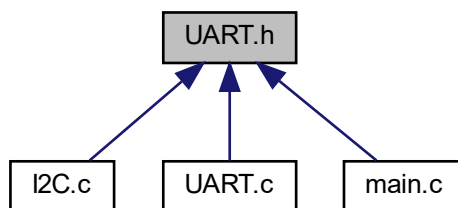
## 6.21 UART.h-Dateireferenz

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

Include-Abhängigkeitsdiagramm für UART.h:



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



### Datenstrukturen

- struct [Buffer](#)

### Makrodefinitionen

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

### Funktionen

- void [initUART](#) (void)
- int16\_t [putsUART](#) (const char \*str)
- int16\_t [getcFIFO\\_TX](#) (volatile uint16\_t \*c)
- int16\_t [putcFIFO\\_TX](#) (char c)

## Variablen

- [Buffer FIFO](#)

### 6.21.1 Makro-Dokumentation

#### 6.21.1.1 BAUDRATE

```
#define BAUDRATE 9600
```

Definiert in Zeile [18](#) der Datei [UART.h](#).

#### 6.21.1.2 BRGVAL

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

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

### 6.21.2 Dokumentation der Funktionen

#### 6.21.2.1 getcFIFO\_TX()

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

Definiert in Zeile [165](#) der Datei [main\\_less.c](#).

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



### 6.21.2.2 initUART()

```
void initUART (  
    void )
```

Definiert in Zeile 100 der Datei [main\\_less.c](#).

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



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

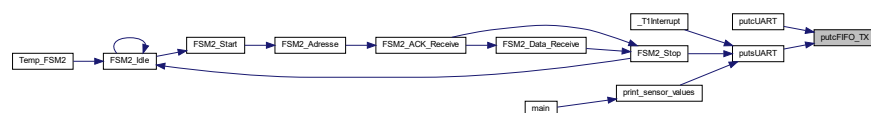


### 6.21.2.3 putcFIFO\_TX()

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

Definiert in Zeile 147 der Datei [main\\_less.c](#).

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





### 6.21.2.4 putsUART()

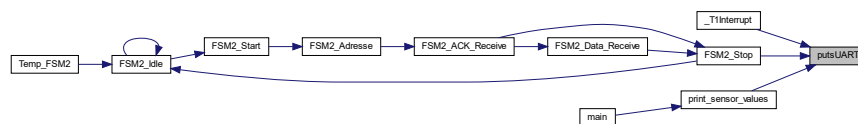
```
int16_t putsUART (
    const char * str )
```

Definiert in Zeile 190 der Datei [main\\_less.c](#).

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



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



## 6.21.3 Variablen-Dokumentation

### 6.21.3.1 FIFO

```
Buffer FIFO [extern]
```

Definiert in Zeile 56 der Datei [main\\_less.c](#).

## 6.22 UART.h

[gehe zur Dokumentation dieser Datei](#)

```
00001
00002 /* Files to Include */
00003
00004 #include "user.h"
00005 #include <stdint.h> /* Includes uint16_t definition */
00006 #include <stdbool.h> /* Includes true/false definition */
00007 #include <string.h>
00008 #include <stdio.h>
00009 #include <stdlib.h>
00010
00011 #include <xc.h>
00012
00013
```

```

00014
00015  /* Konstanten                                     */
00016
00018 #define BAUDRATE 9600
00019 #define BRGVAL ((FCY/BAUDRATE)/16)-1
00020
00021
00022  /* Typedef                                         */
00023
00025 typedef struct
00026 {
00027     uint8_t data[BUFFER_SIZE];
00028     uint8_t read; // zeigt auf das Feld mit dem ältesten Inhalt
00029     uint8_t write; // zeigt immer auf leeres Feld
00030 }Buffer;
00031
00032 #ifdef MAIN
00033
00034
00035  /* Globale Variable Declaration                   */
00036
00038 Buffer FIFO = {{}, 0, 0}; //FIFO zum Versenden über UART
00039 #else
00040 extern Buffer FIFO;
00041 #endif
00042
00043
00044  /* Prototypen                                     */
00045
00047 void initUART(void);
00048
00049 int16_t putsUART(const char *str);
00050 int16_t getcFIFO_TX(volatile uint16_t *c);
00051 //int16_t getcFIFO_RX(char *c);
00052
00053 int16_t putcFIFO_TX(char c);
00054 //int16_t putcFIFO_RX(char c);

```

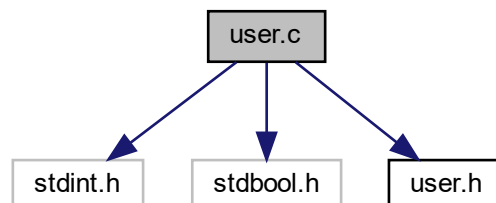
## 6.23 user.c-Dateireferenz

```

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

```

Include-Abhängigkeitsdiagramm für user.c:



## Funktionen

- void `InitApp` (void)
- void `setLED` (uint16\_t nr)

## 6.23.1 Dokumentation der Funktionen

### 6.23.1.1 InitApp()

```
void InitApp (
    void )
```

Definiert in Zeile 26 der Datei `user.c`.

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



### 6.23.1.2 setLED()

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

Definiert in Zeile 35 der Datei `user.c`.

## 6.24 user.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 #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
00023 /* <Initialize variables in user.h and insert code for user algorithms.> */
00024
00025
```

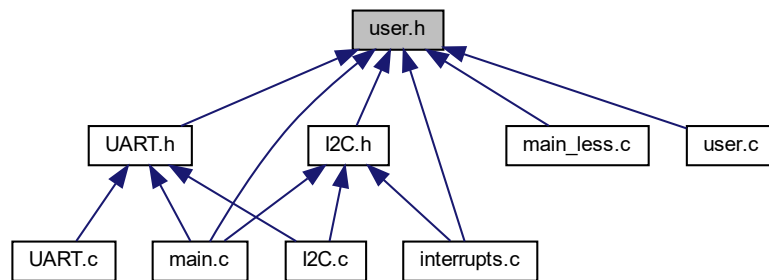
```

00026 void InitApp(void)
00027 {
00028     /* TODO Initialize User Ports/Peripherals/Project here */
00029     /* Setup analog functionality and port direction */
00030     /* Initialize peripherals */
00031 }
00032 void setLED(uint16_t nr)
00033 {
00034     if (nr>=4) return;
00035     LATB = LATB | (1 << (nr+8));
00036 }
00037 //4-bit Wort -> RB8-11
00038 //uint16_t leds=0b 0000 0000 0000 1101;
00039 //LATB = (LATB & ~0b0000111100000000) | ((leds<<8) &0b0000111100000000);

```

## 6.25 user.h-Dateireferenz

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



### Makrodefinitionen

- `#define LED0_LATB8`
- `#define LED1_LATB9`
- `#define LED2_LATB10`
- `#define LED3_LATB11`
- `#define T0 !_RG12`
- `#define T1 !_RG13`
- `#define T2 !_RG14`
- `#define T3 !_RG15`
- `#define BUFFER_FAIL 0`
- `#define BUFFER_SUCCESS 1`
- `#define BUFFER_SIZE 256`
- `#define SENSOR_TIME 1`

### Funktionen

- void `InitApp` (void)

## 6.25.1 Makro-Dokumentation

### 6.25.1.1 BUFFER\_FAIL

```
#define BUFFER_FAIL 0
```

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

### 6.25.1.2 BUFFER\_SIZE

```
#define BUFFER_SIZE 256
```

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

### 6.25.1.3 BUFFER\_SUCCESS

```
#define BUFFER_SUCCESS 1
```

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

### 6.25.1.4 LED0

```
#define LED0 _LATB8
```

Definiert in Zeile 4 der Datei [user.h](#).

### 6.25.1.5 LED1

```
#define LED1 _LATB9
```

Definiert in Zeile 5 der Datei [user.h](#).

#### 6.25.1.6 LED2

```
#define LED2 _LATB10
```

Definiert in Zeile 6 der Datei [user.h](#).

#### 6.25.1.7 LED3

```
#define LED3 _LATB11
```

Definiert in Zeile 7 der Datei [user.h](#).

#### 6.25.1.8 SENSOR\_TIME

```
#define SENSOR_TIME 1
```

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

#### 6.25.1.9 T0

```
#define T0 !_RG12
```

Definiert in Zeile 8 der Datei [user.h](#).

#### 6.25.1.10 T1

```
#define T1 !_RG13
```

Definiert in Zeile 9 der Datei [user.h](#).

#### 6.25.1.11 T2

```
#define T2 !_RG14
```

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

### 6.25.1.12 T3

```
#define T3 !_RG15
```

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

## 6.25.2 Dokumentation der Funktionen

### 6.25.2.1 InitApp()

```
void InitApp (
    void )
```

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

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



## 6.26 user.h

[gehe zur Dokumentation dieser Datei](#)

```
00001
00002 /* User Level #define Macros */
00003
00004 #define LED0 _LATB8
00005 #define LED1 _LATB9
00006 #define LED2 _LATB10
00007 #define LED3 _LATB11
00008 #define T0 !_RG12
00009 #define T1 !_RG13
00010 #define T2 !_RG14
00011 #define T3 !_RG15
00012
00013 #define BUFFER_FAIL 0
00014 #define BUFFER_SUCCESS 1
00015 #define BUFFER_SIZE 256
00016 #define SENSOR_TIME 1
00017 /* TODO Application specific user parameters used in user.c may go here */
00018
00019
00020 /* User Function Prototypes */
00021
00022 /* TODO User level functions prototypes (i.e. InitApp) go here */
00023
00024
00025 void InitApp(void); /* I/O and Peripheral Initialization */
00026
00027
```

## 6.27 Code Style.markdown-Dateireferenz

## 6.28 Dokumentation.markdown-Dateireferenz





# Index

- [\\_AddressError](#)
    - [traps.c, 78](#)
  - [\\_DefaultInterrupt](#)
    - [traps.c, 78](#)
  - [\\_MathError](#)
    - [traps.c, 78](#)
  - [\\_OscillatorFail](#)
    - [traps.c, 78](#)
  - [\\_StackError](#)
    - [traps.c, 78](#)
  - [\\_T1Interrupt](#)
    - [interrupts.c, 48](#)
    - [main\\_less.c, 54](#)
  - [\\_U1TXInterrupt](#)
    - [main\\_less.c, 55](#)
    - [UART.c, 81](#)
- [address](#)
  - [I2C\\_struct, 12](#)
- [BAUDRATE](#)
  - [main\\_less.c, 52](#)
  - [UART.h, 87](#)
- [BRGVAL](#)
  - [main\\_less.c, 53](#)
  - [UART.h, 87](#)
- [Buffer, 9](#)
  - [data, 9](#)
  - [read, 10](#)
  - [write, 10](#)
- [BUFFER\\_FAIL](#)
  - [main\\_less.c, 53](#)
  - [user.h, 93](#)
- [Buffer\\_I2C\\_FSM, 10](#)
  - [data, 11](#)
  - [read, 11](#)
  - [write, 11](#)
- [BUFFER\\_SIZE](#)
  - [main\\_less.c, 53](#)
  - [user.h, 93](#)
- [BUFFER\\_SUCCESS](#)
  - [main\\_less.c, 53](#)
  - [user.h, 93](#)
- [Code Style.markdown, 95](#)
- [configuration\\_bits.c, 15](#)
- [ConfigureOscillator](#)
  - [system.c, 70](#)
  - [system.h, 75](#)
- [data](#)
  - [Buffer, 9](#)
  - [Buffer\\_I2C\\_FSM, 11](#)
  - [main\\_less.c, 64](#)
- [DELAY\\_ANPASSUNG](#)
  - [main\\_less.c, 64](#)
  - [system.h, 76](#)
- [delay\\_ms](#)
  - [main\\_less.c, 55](#)
  - [system.c, 71](#)
  - [system.h, 75](#)
- [dol2C](#)
  - [I2C.c, 17](#)
  - [I2C.h, 35](#)
- [Dokumentation.markdown, 95](#)
- [Error](#)
  - [I2C.h, 35](#)
- [exchangeI2C](#)
  - [I2C.c, 18](#)
  - [I2C.h, 35](#)
- [FCY](#)
  - [system.h, 74](#)
- [FIFO](#)
  - [main\\_less.c, 65](#)
  - [UART.h, 89](#)
- [FIFO\\_I2C](#)
  - [I2C.h, 45](#)
- [Finished](#)
  - [I2C.h, 35](#)
- [FSM2\\_ACK\\_Receive](#)
  - [main\\_less.c, 56](#)
- [FSM2\\_Adresse](#)
  - [main\\_less.c, 56](#)
- [FSM2\\_Data\\_Receive](#)
  - [main\\_less.c, 57](#)
- [FSM2\\_Idle](#)
  - [main\\_less.c, 58](#)
- [FSM2\\_Start](#)
  - [main\\_less.c, 58](#)
- [FSM2\\_Stop](#)
  - [main\\_less.c, 59](#)
- [FSM\\_Adresse\\_Read](#)
  - [I2C.c, 19](#)
  - [I2C.h, 37](#)
- [FSM\\_Adresse\\_Write](#)
  - [I2C.c, 20](#)
  - [I2C.h, 38](#)
- [FSM\\_Idle](#)
  - [I2C.c, 21](#)

- I2C.h, [39](#)
- FSM\_RECV\_EN
  - I2C.c, [22](#)
  - I2C.h, [40](#)
- FSM\_Repeated\_Start
  - I2C.c, [23](#)
  - I2C.h, [41](#)
- FSM\_Start
  - I2C.c, [23](#)
  - I2C.h, [42](#)
- FSM\_Stop
  - I2C.c, [24](#)
  - I2C.h, [42](#)
- get\_I2C\_struct\_FIFO
  - I2C.c, [25](#)
- getcFIFO\_TX
  - main\_less.c, [59](#)
  - UART.c, [82](#)
  - UART.h, [87](#)
- HEARTBEAT\_MS
  - main.c, [49](#)
  - main\_less.c, [53](#)
- I2C.c, [16](#), [28](#)
  - dol2C, [17](#)
  - exchangel2C, [18](#)
  - FSM\_Adresse\_Read, [19](#)
  - FSM\_Adresse\_Write, [20](#)
  - FSM\_Idle, [21](#)
  - FSM\_RECV\_EN, [22](#)
  - FSM\_Repeated\_Start, [23](#)
  - FSM\_Start, [23](#)
  - FSM\_Stop, [24](#)
  - get\_I2C\_struct\_FIFO, [25](#)
  - initI2C, [26](#)
  - print\_sensor\_values, [26](#)
  - put\_I2C\_struct\_FIFO, [27](#)
- I2C.h, [32](#), [46](#)
  - dol2C, [35](#)
  - Error, [35](#)
  - exchangel2C, [35](#)
  - FIFO\_I2C, [45](#)
  - Finished, [35](#)
  - FSM\_Adresse\_Read, [37](#)
  - FSM\_Adresse\_Write, [38](#)
  - FSM\_Idle, [39](#)
  - FSM\_RECV\_EN, [40](#)
  - FSM\_Repeated\_Start, [41](#)
  - FSM\_Start, [42](#)
  - FSM\_Stop, [42](#)
  - I2C\_SCL, [33](#)
  - I2C\_SCL\_TRIS, [34](#)
  - I2C\_SDA, [34](#)
  - I2C\_SDA\_TRIS, [34](#)
  - i2c\_status\_t, [34](#)
  - I2C\_test\_struct, [45](#)
  - initI2C, [43](#)
  - Pending, [35](#)
  - print\_sensor\_values, [44](#)
  - read\_data\_buffer\_light, [45](#)
  - read\_data\_buffer\_temp, [45](#)
  - StateFunc, [34](#)
  - status\_licht, [45](#)
  - status\_temperatur, [45](#)
  - trigger\_FSM, [45](#)
  - write\_data\_buffer\_light, [45](#)
  - write\_data\_buffer\_temp, [46](#)
- I2C\_SCL
  - I2C.h, [33](#)
  - main\_less.c, [53](#)
- I2C\_SCL\_TRIS
  - I2C.h, [34](#)
  - main\_less.c, [54](#)
- I2C\_SDA
  - I2C.h, [34](#)
  - main\_less.c, [54](#)
- I2C\_SDA\_TRIS
  - I2C.h, [34](#)
  - main\_less.c, [54](#)
- i2c\_status\_t
  - I2C.h, [34](#)
- I2C\_struct, [12](#)
  - address, [12](#)
  - num\_read, [12](#)
  - num\_write, [13](#)
  - readbuf, [13](#)
  - status, [13](#)
  - writebuf, [13](#)
- I2C\_test\_struct
  - I2C.h, [45](#)
- init\_ms\_t4
  - main\_less.c, [60](#)
  - system.c, [71](#)
  - system.h, [76](#)
- init\_timer1
  - main\_less.c, [60](#)
  - system.c, [72](#)
  - system.h, [76](#)
- InitApp
  - user.c, [91](#)
  - user.h, [95](#)
- initI2C
  - I2C.c, [26](#)
  - I2C.h, [43](#)
  - main\_less.c, [61](#)
- initUART
  - main\_less.c, [61](#)
  - UART.c, [82](#)
  - UART.h, [87](#)
- interrupts.c, [47](#), [48](#)
  - \_T1Interrupt, [48](#)
- LED0
  - user.h, [93](#)
- LED1
  - user.h, [93](#)

LED2  
  user.h, 93

LED3  
  user.h, 94

MAIN  
  main.c, 49

main  
  main.c, 50  
  main\_less.c, 62

main.c, 49, 50  
  HEARTBEAT\_MS, 49  
  MAIN, 49  
  main, 50

main\_less.c, 51, 65  
  \_T1Interrupt, 54  
  \_U1TXInterrupt, 55  
  BAUDRATE, 52  
  BRGVAL, 53  
  BUFFER\_FAIL, 53  
  BUFFER\_SIZE, 53  
  BUFFER\_SUCCESS, 53  
  data, 64  
  DELAY\_ANPASSUNG, 64  
  delay\_ms, 55  
  FIFO, 65  
  FSM2\_ACK\_Receive, 56  
  FSM2\_Adresse, 56  
  FSM2\_Data\_Receive, 57  
  FSM2\_Idle, 58  
  FSM2\_Start, 58  
  FSM2\_Stop, 59  
  getcFIFO\_TX, 59  
  HEARTBEAT\_MS, 53  
  I2C\_SCL, 53  
  I2C\_SCL\_TRIS, 54  
  I2C\_SDA, 54  
  I2C\_SDA\_TRIS, 54  
  init\_ms\_t4, 60  
  init\_timer1, 60  
  initI2C, 61  
  initUART, 61  
  main, 62  
  putcFIFO\_TX, 62  
  putcUART, 63  
  putsUART, 63  
  StateFunc, 54  
  Temp\_FSM2, 64

num\_read  
  I2C\_struct, 12

num\_write  
  I2C\_struct, 13

Pending  
  I2C.h, 35

print\_sensor\_values  
  I2C.c, 26  
  I2C.h, 44

put\_I2C\_struct\_FIFO  
  I2C.c, 27

putcFIFO\_TX  
  main\_less.c, 62  
  UART.c, 83  
  UART.h, 88

putcUART  
  main\_less.c, 63  
  UART.c, 83

putsUART  
  main\_less.c, 63  
  UART.c, 83  
  UART.h, 88

read  
  Buffer, 10  
  Buffer\_I2C\_FSM, 11

read\_data\_buffer\_light  
  I2C.h, 45

read\_data\_buffer\_temp  
  I2C.h, 45

readbuf  
  I2C\_struct, 13

SENSOR\_TIME  
  user.h, 94

setLED  
  user.c, 91

StateFunc  
  I2C.h, 34  
  main\_less.c, 54

status  
  I2C\_struct, 13

status\_licht  
  I2C.h, 45

status\_temperatur  
  I2C.h, 45

SYS\_FREQ  
  system.h, 74

system.c, 70, 72  
  ConfigureOscillator, 70  
  delay\_ms, 71  
  init\_ms\_t4, 71  
  init\_timer1, 72

system.h, 74, 77  
  ConfigureOscillator, 75  
  DELAY\_ANPASSUNG, 76  
  delay\_ms, 75  
  FCY, 74  
  init\_ms\_t4, 76  
  init\_timer1, 76  
  SYS\_FREQ, 74

T0  
  user.h, 94

T1  
  user.h, 94

T2  
  user.h, 94

- T3
  - user.h, [94](#)
- Temp\_FSM2
  - main\_less.c, [64](#)
- traps.c, [77](#), [79](#)
  - \_AddressError, [78](#)
  - \_DefaultInterrupt, [78](#)
  - \_MathError, [78](#)
  - \_OscillatorFail, [78](#)
  - \_StackError, [78](#)
- trigger\_FSM
  - I2C.h, [45](#)
- UART.c, [81](#), [84](#)
  - \_U1TXInterrupt, [81](#)
  - getcFIFO\_TX, [82](#)
  - initUART, [82](#)
  - putcFIFO\_TX, [83](#)
  - putcUART, [83](#)
  - putsUART, [83](#)
- UART.h, [86](#), [89](#)
  - BAUDRATE, [87](#)
  - BRGVAL, [87](#)
  - FIFO, [89](#)
  - getcFIFO\_TX, [87](#)
  - initUART, [87](#)
  - putcFIFO\_TX, [88](#)
  - putsUART, [88](#)
- user.c, [90](#), [91](#)
  - InitApp, [91](#)
  - setLED, [91](#)
- user.h, [92](#), [95](#)
  - BUFFER\_FAIL, [93](#)
  - BUFFER\_SIZE, [93](#)
  - BUFFER\_SUCCESS, [93](#)
  - InitApp, [95](#)
  - LED0, [93](#)
  - LED1, [93](#)
  - LED2, [93](#)
  - LED3, [94](#)
  - SENSOR\_TIME, [94](#)
  - T0, [94](#)
  - T1, [94](#)
  - T2, [94](#)
  - T3, [94](#)
- write
  - Buffer, [10](#)
  - Buffer\_I2C\_FSM, [11](#)
- write\_data\_buffer\_light
  - I2C.h, [45](#)
- write\_data\_buffer\_temp
  - I2C.h, [46](#)
- writebuf
  - I2C\_struct, [13](#)