Project_Template

Erzeugt von Doxygen 1.9.3

Kapitel 1

Embedded1 und Embedded2 Aufgabendokumentation

p_formatierung

1.1 Programmdokumentation

Die Dokumentation erfolgt gtrennt nach den jeweiligen Aufgaben

1.1.1 Aufgabe 1

- die Main Funktion - main()
- Systemfrequenz
  - Die Oscillatoreinstellungen sind in ConfigureOscillator() programmiert
  - Das Makrodefine #SYS_FREQ wird dabei ausgewertet
- FÃ¼r die GPIOs wie Taster und LEDs sind in der Datei user.h diverse Zugriffs Makros definiert sowie eine Initialisierungsfunktion initApp()
- Softwaredelay Funktion delay_ms()

1.1.2 Aufgabe 2 und 3

- Delayfunktion mit dem Timer 1 delay_ms_t1() und zugehÃ¶rige Initialisierung init_ms_t1()
- Die LED und Taster Tasks: TaskLED0() und TaskTaster0()

1.1.3 Restliche Aufgaben

- FÃ¼r die Prozessorauslastung: Auslastung() und init_T23_gated(void)
- Betriebszeit: initBetriebszeit() setBetriebszeit() getBetriebszeit() _T1Interrupt()
- UART putcUART() putsUART() getcUART()
  - FIFO: getcTX_FIFO() putcTX_FIFO() _U1TXInterrupt() _U1RXInterrupt
- I2C und Temperatursensor: readTemperatureSimpleBlocking() readTemperatureFSM()
- State Machines: diverse Umsetzungen in aFSM.h
- Funktionen fÃ¼r das LC Display in LCD.h
- Dokumentation doku_page_timer_h in uTimer.h

Kapitel 2

Datenstruktur-Verzeichnis

2.1 Datenstrukturen

Hier folgt die Aufzählung aller Datenstrukturen mit einer Kurzbeschreibung:

Kapitel 3

Datei-Verzeichnis

3.1 Auflistung der Dateien

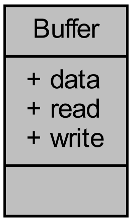
Hier folgt die Aufzählung aller dokumentierten Dateien mit einer Kurzbeschreibung:

Kapitel 4

Datenstruktur-Dokumentation

4.1 Buffer Strukturreferenz

Zusammengehörigkeiten von Buffer:



Datenfelder

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

4.1.1 Ausführliche Beschreibung

Definiert in Zeile 39 der Datei main.c.

4.1.2 Dokumentation der Felder

4.1.2.1 data

```
uint8_t data
```

Definiert in Zeile 40 der Datei main.c.

4.1.2.2 read

```
uint8_t read
```

Definiert in Zeile 41 der Datei main.c.

4.1.2.3 write

```
uint8_t write
```

Definiert in Zeile 42 der Datei main.c.

Die Dokumentation für diese Struktur wurde erzeugt aufgrund der Dateien:

- main.c
- main_less.c

Kapitel 5

Datei-Dokumentation

5.1 configuration_bits.o.d

```
build/XC16_dsPIC33EP512MU810/production/configuration_bits.o: \
 configuration_bits.c
```

5.2 interrupts.o.d

```
build/XC16_dsPIC33EP512MU810/production/interrupts.o: interrupts.c
```

5.3 main.o.d

```
build/XC16_dsPIC33EP512MU810/production/main.o: main.c system.h user.h

system.h:

user.h:
```

5.4 system.o.d

```
build/XC16_dsPIC33EP512MU810/production/system.o: system.c system.h

system.h:
```

5.5 traps.o.d

```
build/XC16_dsPIC33EP512MU810/production/traps.o: traps.c
```

5.6 user.o.d

```
build/XC16_dsPIC33EP512MU810/production/user.o: user.c user.h

user.h:
```

5.7 configuration_bits.c

```

// DSPIC33EP512MU810 Configuration Bit Settings

// 'C' source line config statements

// FGS
#pragma config GWRP = OFF               // General Segment Write-Protect bit (General Segment may be written)
#pragma config GSS = OFF                // General Segment Code-Protect bit (General Segment Code protect is disabled)
#pragma config GSSK = OFF               // General Segment Key bits (General Segment Write Protection and Code Protection is Disabled)

// FOSCSEL
#pragma config FNOSC = FRCDIVN          // Initial Oscillator Source Selection Bits (Internal Fast RC (FRC) Oscillator with postscaler)
#pragma config IESO = ON                // Two-speed Oscillator Start-up Enable bit (Start up device with FRC, then switch to user-selected oscillator source)

// FOSC
#pragma config POSCMD = XT              // Primary Oscillator Mode Select bits (XT Crystal Oscillator Mode)
#pragma config OSCIOFNC = OFF           // OSC2 Pin Function bit (OSC2 is clock output)
#pragma config IOL1WAY = ON             // Peripheral pin select configuration (Allow only one reconfiguration)
#pragma config FCKSM = CSECMD           // Clock Switching Mode bits (Clock switching is enabled,Fail-safe Clock Monitor is disabled)

// FWDT
#pragma config WDTPOST = PS32768        // Watchdog Timer Postscaler Bits (1:32,768)
#pragma config WDTPRE = PR128           // Watchdog Timer Prescaler bit (1:128)
#pragma config PLLKEN = ON              // PLL Lock Wait Enable bit (Clock switch to PLL source will wait until the PLL lock signal is valid.)
#pragma config WINDIS = OFF             // Watchdog Timer Window Enable bit (Watchdog Timer in Non-Window mode)
#pragma config FWDTEN = ON              // Watchdog Timer Enable bit (Watchdog timer always enabled)

// FPOR
#pragma config FPWRT = PWR128           // Power-on Reset Timer Value Select bits (128ms)
#pragma config BOREN = ON               // Brown-out Reset (BOR) Detection Enable bit (BOR is enabled)
#pragma config ALTI2C1 = OFF            // Alternate I2C pins for I2C1 (SDA1/SCK1 pins are selected as the I/O pins for I2C1)
#pragma config ALTI2C2 = ON            // Alternate I2C pins for I2C2 (SDA2/SCK2 pins are selected as the I/O pins for I2C2)

// FICD
#pragma config ICS = PGD1               // ICD Communication Channel Select bits (Communicate on PGEC1 and PGED1)
#pragma config RSTPRI = PF              // Reset Target Vector Select bit (Device will obtain reset instruction from Primary flash)
#pragma config JTAGEN = OFF             // JTAG Enable bit (JTAG is disabled)

// FAS
#pragma config AWRP = OFF               // Auxiliary Segment Write-protect bit (Auxiliary program memory is not write-protected)
#pragma config APL = OFF                // Auxiliary Segment Code-protect bit (Aux Flash Code protect is disabled)
#pragma config APLK = OFF               // Auxiliary Segment Key bits (Aux Flash Write Protection and Code Protection is Disabled)

// #pragma config statements should precede project file includes.
// Use project enums instead of #define for ON and OFF.

#include <xc.h>




```

5.8 interrupts.c

```

/* Files to Include                                                           */

/* Device header file */
#if defined(__XC16__)
    #include <xc.h>
#elif defined(__C30__)
    #if defined(__dsPIC33E__)
        #include <p33Exxxx.h>
    #elif defined(__dsPIC33F__)
```

```
        #include <p33Fxxxx.h>
    #endif
#endif

#include <stdint.h>        /* Includes uint16_t definition   */
#include <stdbool.h>       /* Includes true/false definition */


/* Interrupt Vector Options                                                   */

/*                                                                            */
/* Refer to the C30 (MPLAB C Compiler for PIC24F MCUs and dsPIC33F DSCs) User */
/* Guide for an up to date list of the available interrupt options.           */
/* Alternately these names can be pulled from the device linker scripts.      */
/*                                                                            */
/* dsPIC33F Primary Interrupt Vector Names:                                   */
/*                                                                            */
/* _INT0Interrupt      _C1Interrupt                                           */
/* _IC1Interrupt       _DMA3Interrupt                                         */
/* _OC1Interrupt       _IC3Interrupt                                          */
/* _T1Interrupt        _IC4Interrupt                                          */
/* _DMA0Interrupt      _IC5Interrupt                                          */
/* _IC2Interrupt       _IC6Interrupt                                          */
/* _OC2Interrupt       _OC5Interrupt                                          */
/* _T2Interrupt        _OC6Interrupt                                          */
/* _T3Interrupt        _OC7Interrupt                                          */
/* _SPI1ErrInterrupt   _OC8Interrupt                                          */
/* _SPI1Interrupt      _DMA4Interrupt                                         */
/* _U1RXInterrupt      _T6Interrupt                                           */
/* _U1TXInterrupt      _T7Interrupt                                           */
/* _ADC1Interrupt      _SI2C2Interrupt                                        */
/* _DMA1Interrupt      _MI2C2Interrupt                                        */
/* _SI2C1Interrupt     _T8Interrupt                                           */
/* _MI2C1Interrupt     _T9Interrupt                                           */
/* _CNInterrupt        _INT3Interrupt                                         */
/* _INT1Interrupt      _INT4Interrupt                                         */
/* _ADC2Interrupt      _C2RxRdyInterrupt                                      */
/* _DMA2Interrupt      _C2Interrupt                                           */
/* _OC3Interrupt       _DCIErrInterrupt                                       */
/* _OC4Interrupt       _DCIInterrupt                                          */
/* _T4Interrupt        _DMA5Interrupt                                         */
/* _T5Interrupt        _U1ErrInterrupt                                        */
/* _INT2Interrupt      _U2ErrInterrupt                                        */
/* _U2RXInterrupt      _DMA6Interrupt                                         */
/* _U2TXInterrupt      _DMA7Interrupt                                         */
/* _SPI2ErrInterrupt   _C1TxReqInterrupt                                      */
/* _SPI2Interrupt      _C2TxReqInterrupt                                      */
/* _C1RxRdyInterrupt                                                          */
/*                                                                            */
/* dsPIC33E Primary Interrupt Vector Names:                                   */
/*                                                                            */
/* _INT0Interrupt     _IC4Interrupt      _U4TXInterrupt                       */
/* _IC1Interrupt      _IC5Interrupt      _SPI3ErrInterrupt                    */
/* _OC1Interrupt      _IC6Interrupt      _SPI3Interrupt                       */
/* _T1Interrupt       _OC5Interrupt      _OC9Interrupt                        */
/* _DMA0Interrupt     _OC6Interrupt      _IC9Interrupt                        */
/* _IC2Interrupt      _OC7Interrupt      _PWM1Interrupt                       */
/* _OC2Interrupt      _OC8Interrupt      _PWM2Interrupt                       */
/* _T2Interrupt       _PMPInterrupt      _PWM3Interrupt                       */
/* _T3Interrupt       _DMA4Interrupt     _PWM4Interrupt                       */
/* _SPI1ErrInterrupt  _T6Interrupt       _PWM5Interrupt                       */
/* _SPI1Interrupt     _T7Interrupt       _PWM6Interrupt                       */
/* _U1RXInterrupt     _SI2C2Interrupt    _PWM7Interrupt                       */
/* _U1TXInterrupt     _MI2C2Interrupt    _DMA8Interrupt                       */
/* _AD1Interrupt      _T8Interrupt       _DMA9Interrupt                       */
/* _DMA1Interrupt     _T9Interrupt       _DMA10Interrupt                      */
/* _NVMInterrupt      _INT3Interrupt     _DMA11Interrupt                      */
/* _SI2C1Interrupt    _INT4Interrupt     _SPI4ErrInterrupt                    */
/* _MI2C1Interrupt    _C2RxRdyInterrupt  _SPI4Interrupt                       */
/* _CM1Interrupt      _C2Interrupt       _OC10Interrupt                       */
/* _CNInterrupt       _QEI1Interrupt     _IC10Interrupt                       */
/* _INT1Interrupt     _DCIEInterrupt     _OC11Interrupt                       */
/* _AD2Interrupt      _DCIInterrupt      _IC11Interrupt                       */
/* _IC7Interrupt      _DMA5Interrupt     _OC12Interrupt                       */
/* _IC8Interrupt      _RTCCInterrupt     _IC12Interrupt                       */
/* _DMA2Interrupt     _U1ErrInterrupt    _DMA12Interrupt                      */
/* _OC3Interrupt      _U2ErrInterrupt    _DMA13Interrupt                      */
/* _OC4Interrupt      _CRCInterrupt      _DMA14Interrupt                      */
/* _T4Interrupt       _DMA6Interrupt     _OC13Interrupt                       */
/* _T5Interrupt       _DMA7Interrupt     _IC13Interrupt                       */
/* _INT2Interrupt     _C1TxReqInterrupt  _OC14Interrupt                       */
/* _U2RXInterrupt     _C2TxReqInterrupt  _IC14Interrupt                       */
/* _U2TXInterrupt     _QEI2Interrupt     _OC15Interrupt                       */
/* _SPI2ErrInterrupt  _U3ErrInterrupt    _IC15Interrupt                       */
/* _SPI2Interrupt     _U3RXInterrupt     _OC16Interrupt                       */
/* _C1RxRdyInterrupt  _U3TXInterrupt     _IC16Interrupt                       */
/* _C1Interrupt       _USB1Interrupt     _ICDInterrupt                        */
/* _DMA3Interrupt     _U4ErrInterrupt    _PWMSpEventMatchInterrupt            */
/* _IC3Interrupt      _U4RXInterrupt     _PWMSecSpEventMatchInterrupt         */
/*                                                                            */
/* For alternate interrupt vector naming, simply add ’Alt’ between the prim.  */
/* interrupt vector name ’_’ and the first character of the primary interrupt */
/* vector name.  There is no Alternate Vector or ’AIVT’ for the 33E family.   */
/*                                                                            */
/* For example, the vector name _ADC2Interrupt becomes _AltADC2Interrupt in   */
/* the alternate vector table.                                                */
/*                                                                            */
/* Example Syntax:                                                            */
/*                                                                            */
/* void __attribute__((interrupt,auto_psv)) <Vector Name>(void)               */
/* {                                                                          */
/*     <Clear Interrupt Flag>                                                 */
/* }                                                                          */
/*                                                                            */
/* For more comprehensive interrupt examples refer to the C30 (MPLAB C        */
/* Compiler for PIC24 MCUs and dsPIC DSCs) User Guide in the                  */
/* <C30 compiler instal directory>/doc directory for the latest compiler      */
/* release.  For XC16, refer to the MPLAB XC16 C Compiler User’s Guide in the */
/* <XC16 compiler instal directory>/doc folder.                               */
/*                                                                            */

/* Interrupt Routines                                                         */

/* TODO Add interrupt routine code here. */
```

5.9 main.c

```

/* Files to Include                                                           */

#include <xc.h>

#include <stdint.h>        /* Includes uint16_t definition                    */
#include <stdbool.h>       /* Includes true/false definition                  */
#include <string.h>
#include <stdio.h>
#include <stdlib.h>

#include "system.h"        /* System funct/params, like osc/peripheral config */
#include "user.h"          /* User funct/params, such as InitApp              */


/* Global Variable Declaration                                                */

#define HEARTBEAT_MS 1
uint32_t DELAY_ANPASSUNG;

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

//FIFO
#define BUFFER_FAIL     0
#define BUFFER_SUCCESS  1
#define BUFFER_SIZE 128


//I2C
#define I2C_SCL      _RA2
#define I2C_SDA      _RA3
#define I2C_SCL_TRIS _TRISA2
#define I2C_SDA_TRIS _TRISA3
uint8_t data[2];

/*Typen-Definitionen***********************************************************/
typedef struct {
  uint8_t data[BUFFER_SIZE];
  uint8_t read; // zeigt auf das Feld mit dem ältesten Inhalt
  uint8_t write; // zeigt immer auf leeres Feld
}Buffer;

Buffer FIFO = {{}, 0, 0};

typedef void *(*StateFunc)();


/*Prototypes*******************************************************************/


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

```c
int16_t getcFIFO_TX(volatile uint16_t *c);
//int16_t getcFIFO_RX(char *c);

int16_t putcFIFO_TX(char c);
//int16_t putcFIFO_RX(char c);

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);

/*Funktionen*******************************************************************/

void delay_ms(uint16_t milliseconds) {
    uint32_t i=0;
    for (i=0;i<(DELAY_ANPASSUNG*(uint32_t)milliseconds);i++){
         }
}

void __attribute__((__interrupt__, no_auto_psv)) _T1Interrupt(void)
    {
        _T1IF = 0; //Clear Timer1 interrupt flag

        putsUART("Hello World\n");

    }

//UART
void initUART(){
    U1MODEbits.STSEL = 0; // 1-Stop bit
    U1MODEbits.PDSEL = 0; // No Parity, 8-Data bits
    U1MODEbits.ABAUD = 0; // Auto-Baud disabled
    U1MODEbits.UEN = 0;
    U1MODEbits.LPBACK = 0;
    U1MODEbits.RXINV = 0;
    //U1MODEbits.ALTIO = 0;

    U1MODEbits.URXINV = 0;
    U1MODEbits.RTSMD = 0;

    U1MODEbits.BRGH = 0; // Standard-Speed mode
    U1BRG = BRGVAL; // Baud Rate setting for 9600

    U1STAbits.UTXISEL0 = 0; // Interrupt after one TX character is transmitted
    U1STAbits.UTXISEL1 = 0;
    U1STAbits.UTXBRK = 0;
    U1STAbits.ADDEN = 0;
    U1STAbits.UTXINV = 0;
    U1STAbits.URXISEL = 0;
    U1STA = U1STA | 0b0001000000000000;
    //_URXEN = 1;

    //_U1RXIE = 1; // Enable UART RX interrupt

    U1MODEbits.UARTEN = 1; // Enable UART
    //delay_ms(2);
    U1STAbits.UTXEN = 1; // Enable UART TX

    /* Wait at least 105 microseconds (1/9600) before sending first char */
    delay_ms(2);
    _U1TXIE = 1; // Enable UART TX interrupt

}

void __attribute__((__interrupt__, no_auto_psv)) _U1TXInterrupt(void)
{
    _U1TXIF = 0; // Clear TX Interrupt flag

    getcFIFO_TX(&U1TXREG);

}




int16_t putcFIFO_TX(char c)
{
  //if (buffer.write >= BUFFER_SIZE)
  //  buffer.write = 0; // erhöht sicherheit
  _LATF0 = 1;
  if ( ( FIFO.write + 1 == FIFO.read ) ||
       ( FIFO.read == 0 && FIFO.write + 1 == BUFFER_SIZE ) )
    return BUFFER_FAIL; // voll

  FIFO.data[FIFO.write] = c;

  FIFO.write++;
  if (FIFO.write >= BUFFER_SIZE)
    FIFO.write = 0;

  return BUFFER_SUCCESS;
}

int16_t getcFIFO_TX(volatile uint16_t *c)
{
  _LATF0 = 1;
  if (FIFO.read == FIFO.write)
    return BUFFER_FAIL;

  *c = FIFO.data[FIFO.read];

  FIFO.read++;
  if (FIFO.read >= BUFFER_SIZE)
    FIFO.read = 0;

  return BUFFER_SUCCESS;
}

int16_t putcUART(char c){
    _LATF0 = 1;
    _GIE = 0; // Interrupts ausschalten
    int16_t erfolg = putcFIFO_TX(c);
    _GIE = 1;
    return erfolg;


}

int16_t putsUART(const char *str) {
    _LATF0 = 1;
    uint16_t i;
    uint16_t length = strlen(str);

    _GIE = 0;   //Global Interrupt disable
    for(i = 0; i < length; i++) {
        //uint16_t ret = putcFIFO_TX(str[i]);
        if(! putcFIFO_TX(str[i]))
            break;
    }
    _GIE = 1;
    int16_t erfolg = -i;
    if(erfolg == -length)
        erfolg *= -1;
    _U1TXIF = 1; //Interuppt Routine Starten um FIFO-Inhalt zu senden
    return erfolg;
}

//I2C

void Temp_FSM2(void)
{
     static StateFunc statefunc = FSM2_Idle;

     statefunc = (StateFunc)(*statefunc)();
}

void initI2C(){
    I2C2CONbits.A10M = 0;
    I2C2BRG = 245; //100kHz

    // Einschalten I2C mit eigenem Workaround, I2C Peripheriemodul kann hier leider nicht verwendet werden
    I2C_SDA_TRIS = 1;    // Pins wie einen Open-Kollektor-Treiber verwenden, d.h. 1 - hochohmig, 0 wird getrieben
    I2C_SCL_TRIS = 1;
    I2C_SDA = 0;
    I2C_SCL = 0;

        int j;
        for (j=0; j<=9; j++)    // takten bis min 1 Byte
        {
          I2C_SCL_TRIS = 0; delay_ms(1); // 5 us wÃ¤ren ausreichend ...100 kBaud
          I2C_SCL_TRIS = 1; delay_ms(1);
        }
        // Start Condition senden
        I2C_SCL_TRIS = 0;  delay_ms(1);
        I2C_SDA_TRIS = 0;  delay_ms(1);
        // Stop Condition senden
        I2C_SCL_TRIS = 1;  delay_ms(1);
        I2C_SDA_TRIS = 1;  delay_ms(1);

        // Nun I2C erst anschalten
        _MI2C2IF = 0; //Interrupt falls noetig
        _MI2C2IE = 0;
        I2C2CONbits.I2CEN = 1;

    //Sensor Pointer auf TEMP Register setzten
    I2C2CONbits.SEN=1; //start
    while(I2C2CONbits.SEN==1){}

    //Tx Device address + Write bit
    I2C2TRN=0b10010000;
    while(I2C2STATbits.TRSTAT==1){}

    if (I2C2STATbits.ACKSTAT==1){   //if NACK received, generate stop condition and exit
        I2C2STATbits.ACKSTAT=0;
        I2C2CONbits.PEN=1;
        while(I2C2CONbits.PEN==1){} //wait for the stop interrupt;
        return;
    }

    //Tx Register Address
    I2C2TRN=0b00000000; //Pointer auf TEMP REGISTER setzten
    while(I2C2STATbits.TRSTAT==1){}

    if (I2C2STATbits.ACKSTAT==1){   //if NACK received, generate stop condition and exit
        I2C2STATbits.ACKSTAT=0;
        I2C2CONbits.PEN=1;
        while(I2C2CONbits.PEN==1){} //wait for the stop interrupt;
        return;
    }

    I2C2CONbits.PEN=1; //stop
    while(I2C2CONbits.PEN==1){} //wait for the stop interrupt
}



void *FSM2_Idle(void)
{
    static int c = 0;
    if (c>=999){
        c=0;
        return FSM2_Start;
    }
    c++;
    return FSM2_Idle;

}

void *FSM2_Start(void)
{
    I2C2CONbits.SEN=1; //Start
    while(I2C2CONbits.SEN==1){}
    return FSM2_Adresse;
}

void *FSM2_Adresse(void)
{
    //Tx Device address + Read bit
    I2C2TRN=0b10010001;
    while(I2C2STATbits.TRSTAT==1){} //Warten solange übertragen wird
    return FSM2_ACK_Receive;
}

void *FSM2_ACK_Receive(void)
{
    if (I2C2STATbits.ACKSTAT==1){   //if NACK received, generate stop condition and exit
        I2C2STATbits.ACKSTAT=0;
        return FSM2_Stop;
    }
    return FSM2_Data_Receive;
}

void *FSM2_Data_Receive(void)
{
    int N=2; //2 bytes empfangen
    int i;

    for(i=0;i<N;i++){
        I2C2CONbits.RCEN=1; //Empfangen aktivieren
        while(I2C2CONbits.RCEN==1){} //RCEN cleared automatically when SSP1IF goes high

        data[i]=I2C2RCV;

        //ACK sequence
        if (i<N-1){ I2C2CONbits.ACKDT=0; } //jedes byte mit ACK bestätigen
        else {I2C2CONbits.ACKDT=1;} //send NACK if this is the last Byte

        I2C2CONbits.ACKEN=1; //start ack/nack sequence
        while(I2C2CONbits.ACKEN==1){}

    } //end for loop
    return FSM2_Stop;
}

void *FSM2_Stop(void)
{
    I2C2CONbits.PEN=1;
    while(I2C2CONbits.PEN==1){} //wait for the stop interrupt

    double temp = data[0]«8|data[1];
    char str[16];
    sprintf(str,"%f",temp/256);
    putsUART("Temperatur: ");
    putsUART(str);
    putsUART("°C");
    putsUART("\n");

    return FSM2_Idle;
}



/* Main Program                                                                */

int16_t main(void)
{
    DELAY_ANPASSUNG = ((SYS_FREQ/96)*2180ull)/1000000ull; //Berechnung der Delay Anpassung
    uint16_t Count = 0;
    /* Configure the oscillator for the device */
    ConfigureOscillator();
    /* Initialize IO ports and peripherals */
    //InitApp();
    initUART();
    init_timer1();
    init_ms_t4();
    initI2C();


    TRISBbits.TRISB8 = 0; //LED als Ausgang
    ANSELBbits.ANSB8 = 0;

    TRISBbits.TRISB9 = 0; //LED als Ausgang
    ANSELBbits.ANSB9 = 0;

   //Taster als Eingänge
    _TRISG12 = 1;
    //Pull-up Widerstände einschalten
    _CNPUG12 = 1;


    _RP66R = _RPOUT_U1TX; //UART Pin Mapping
    RPINR18bits.U1RXR = 0b1011000;

    while(1)
    {
        if(_T4IF)
        {
            _T4IF=0;
            Count++;
            if (Count >= HEARTBEAT_MS)
            {
                Count = 0;
                Temp_FSM2();
            }

        }
    }
}
```

5.10 main_less.c

```

/* Files to Include                                                            */

/* Device header file */
#if defined(__XC16__)
    #include <xc.h>
#elif defined(__C30__)
    #if defined(__dsPIC33E__)
```

```
        #include <p33Exxxx.h>
    #elif defined(__dsPIC33F__)
        #include <p33Fxxxx.h>
    #endif
#endif


#include <stdint.h>        /* Includes uint16_t definition                    */
#include <stdbool.h>       /* Includes true/false definition                  */
#include <string.h>
#include <stdio.h>
#include <stdlib.h>

#include "system.h"        /* System funct/params, like osc/peripheral config */
#include "user.h"          /* User funct/params, such as InitApp              */


/* Global Variable Declaration                                                */

#define HEARTBEAT_MS 1
//UART

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

#define BUFFER_FAIL     0
#define BUFFER_SUCCESS  1
#define BUFFER_SIZE 128
uint32_t DELAY_ANPASSUNG;

//I2C
#define I2C_SCL      _RA2
#define I2C_SDA      _RA3
#define I2C_SCL_TRIS _TRISA2
#define I2C_SDA_TRIS _TRISA3
uint8_t data[2];

/*Typen-Definitionen**********************************************************/

typedef struct {
  uint8_t data[BUFFER_SIZE];
  uint8_t read; // zeigt auf das Feld mit dem ältesten Inhalt
  uint8_t write; // zeigt immer auf leeres Feld
}Buffer;

Buffer FIFO = {{}, 0, 0};

typedef void *(*StateFunc)();


/*Prototypes******************************************************************/
void init_ms_t4(void);

int16_t putsUART(const char *str);
int16_t getcFIFO_TX(volatile uint16_t *c);
//int16_t getcFIFO_RX(char *c);

int16_t putcFIFO_TX(char c);
//int16_t putcFIFO_RX(char c);

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);

/*Funktionen******************************************************************/
void delay_ms(uint16_t milliseconds) {
    uint32_t i=0;
    for (i=0;i<(DELAY_ANPASSUNG*(uint32_t)milliseconds);i++){
         }
}

void __attribute__((__interrupt__, no_auto_psv)) _T1Interrupt(void)
    {
        _T1IF = 0; //Clear Timer1 interrupt flag

        putsUART("Hello World\n");

    }

//UART
void initUART(){
    U1MODEbits.STSEL = 0; // 1-Stop bit
    U1MODEbits.PDSEL = 0; // No Parity, 8-Data bits
    U1MODEbits.ABAUD = 0; // Auto-Baud disabled
    U1MODEbits.UEN = 0;
    U1MODEbits.LPBACK = 0;
    U1MODEbits.RXINV = 0;
    //U1MODEbits.ALTIO = 0;

    U1MODEbits.URXINV = 0;
    U1MODEbits.RTSMD = 0;

    U1MODEbits.BRGH = 0; // Standard-Speed mode
    U1BRG = BRGVAL; // Baud Rate setting for 9600

    U1STAbits.UTXISEL0 = 0; // Interrupt after one TX character is transmitted
    U1STAbits.UTXISEL1 = 0;
    U1STAbits.UTXBRK = 0;
    U1STAbits.ADDEN = 0;
    U1STAbits.UTXINV = 0;
    U1STAbits.URXISEL = 0;
    U1STA = U1STA | 0b0001000000000000;
    //_URXEN = 1;

    //_U1RXIE = 1; // Enable UART RX interrupt

    U1MODEbits.UARTEN = 1; // Enable UART
    //delay_ms(2);
    U1STAbits.UTXEN = 1; // Enable UART TX

    /* Wait at least 105 microseconds (1/9600) before sending first char */
    delay_ms(2);
    _U1TXIE = 1; // Enable UART TX interrupt

}

void __attribute__((__interrupt__)) _U1TXInterrupt(void)
{
    _U1TXIF = 0; // Clear TX Interrupt flag

    getcFIFO_TX(&U1TXREG);

}




int16_t putcFIFO_TX(char c)
{
  //if (buffer.write >= BUFFER_SIZE)
  //  buffer.write = 0; // erhöht sicherheit
  _LATF0 = 1;
  if ( ( FIFO.write + 1 == FIFO.read ) ||
       ( FIFO.read == 0 && FIFO.write + 1 == BUFFER_SIZE ) )
    return BUFFER_FAIL; // voll

  FIFO.data[FIFO.write] = c;

  FIFO.write++;
  if (FIFO.write >= BUFFER_SIZE)
    FIFO.write = 0;

  return BUFFER_SUCCESS;
}

int16_t getcFIFO_TX(volatile uint16_t *c)
{
  _LATF0 = 1;
  if (FIFO.read == FIFO.write)
    return BUFFER_FAIL;

  *c = FIFO.data[FIFO.read];

  FIFO.read++;
  if (FIFO.read >= BUFFER_SIZE)
    FIFO.read = 0;

  return BUFFER_SUCCESS;
}

int16_t putcUART(char c){
    _LATF0 = 1;
    _GIE = 0; // Interrupts ausschalten
    int16_t erfolg = putcFIFO_TX(c);
    _GIE = 1;
    return erfolg;


}
```

```

int16_t putsUART(const char *str) {
    _LATF0 = 1;
    uint16_t i;
    uint16_t length = strlen(str);

    _GIE = 0;   //Global Interrupt disable
    for(i = 0; i < length; i++) {
        //uint16_t ret = putcFIFO_TX(str[i]);
        if(! putcFIFO_TX(str[i]))
            break;
    }
    _GIE = 1;
    int16_t erfolg = -i;
    if(erfolg == -length)
        erfolg *= -1;
    _U1TXIF = 1; //Interuppt Routine Starten um FIFO-Inhalt zu senden
    return erfolg;
}

//Timer1
void init_timer1(){
    __builtin_write_OSCCONL(0b00000011); //SOSC aktivieren
    T1CONbits.TON = 0; // Disable Timer
    T1CONbits.TCS = 1; // Select external clock
    T1CONbits.TSYNC = 0; // Disable Synchronization
    T1CONbits.TCKPS = 0b00; // Select 1:1 Prescaler
    TMR1 = 0x00; // Clear timer register
    PR1 = 32767; // Load the period value, Quarztakt

    IPC0bits.T1IP = 2; // Set Timer 1 Interrupt Priority Level
    IFS0bits.T1IF = 0; // Clear Timer 1 Interrupt Flag
    IEC0bits.T1IE = 1; // Enable Timer1 interrupt
    T1CONbits.TON = 1; // Start Timer
}

//I2C

void Temp_FSM2(void)
{
     static StateFunc statefunc = FSM2_Idle;

     statefunc = (StateFunc)(*statefunc)();
}

void initI2C(){
    I2C2CONbits.A10M = 0;
    I2C2BRG = 245; //100kHz

    // Einschalten I2C mit eigenem Workaround, I2C Peripheriemodul kann hier leider nicht verwendet werden
    I2C_SDA_TRIS = 1;    // Pins wie einen Open-Kollektor-Treiber verwenden, d.h. 1 - hochohmig, 0 wird getrieben
    I2C_SCL_TRIS = 1;
    I2C_SDA = 0;
    I2C_SCL = 0;

        int j;
        for (j=0; j<=9; j++)     // takten bis min 1 Byte
        {
          I2C_SCL_TRIS = 0; delay_ms(1); // 5 us wÃ¤ren ausreichend ...100 kBaud
          I2C_SCL_TRIS = 1; delay_ms(1);
        }
        // Start Condition senden
        I2C_SCL_TRIS = 0;  delay_ms(1);
        I2C_SDA_TRIS = 0;  delay_ms(1);
        // Stop Condition senden
        I2C_SCL_TRIS = 1;  delay_ms(1);
        I2C_SDA_TRIS = 1;  delay_ms(1);

        // Nun I2C erst anschalten
        _MI2C2IF = 0; //Interrupt falls noetig
        _MI2C2IE = 0;
        I2C2CONbits.I2CEN = 1;

    //Sensor Pointer auf TEMP Register setzten
    I2C2CONbits.SEN=1; //start
    while(I2C2CONbits.SEN==1){}

    //Tx Device address + Write bit
    I2C2TRN=0b10010000;
    while(I2C2STATbits.TRSTAT==1){}

    if (I2C2STATbits.ACKSTAT==1){   //if NACK received, generate stop condition and exit
        I2C2STATbits.ACKSTAT=0;
        I2C2CONbits.PEN=1;
        while(I2C2CONbits.PEN==1){} //wait for the stop interrupt;
        return;
    }

    //Tx Register Address
    I2C2TRN=0b00000000; //Pointer auf TEMP REGISTER setzten
    while(I2C2STATbits.TRSTAT==1){}

    if (I2C2STATbits.ACKSTAT==1){   //if NACK received, generate stop condition and exit
        I2C2STATbits.ACKSTAT=0;
        I2C2CONbits.PEN=1;
        while(I2C2CONbits.PEN==1){} //wait for the stop interrupt;
        return;
    }

    I2C2CONbits.PEN=1; //stop
    while(I2C2CONbits.PEN==1){} //wait for the stop interrupt
}



void *FSM2_Idle(void)
{
    static int c = 0;
    if (c>=999){
        c=0;
        return FSM2_Start;
    }
    c++;
    return FSM2_Idle;

}

void *FSM2_Start(void)
{
    I2C2CONbits.SEN=1; //Start
    while(I2C2CONbits.SEN==1){}
    return FSM2_Adresse;
}

void *FSM2_Adresse(void)
{
    //Tx Device address + Read bit
    I2C2TRN=0b10010001;
    while(I2C2STATbits.TRSTAT==1){} //Warten solange übertragen wird
    return FSM2_ACK_Receive;
}

void *FSM2_ACK_Receive(void)
{
    if (I2C2STATbits.ACKSTAT==1){   //if NACK received, generate stop condition and exit
        I2C2STATbits.ACKSTAT=0;
        return FSM2_Stop;
    }
    return FSM2_Data_Receive;
}

void *FSM2_Data_Receive(void)
{
    int N=2; //2 bytes empfangen
    int i;

    for(i=0;i<N;i++){
        I2C2CONbits.RCEN=1; //Empfangen aktivieren
        while(I2C2CONbits.RCEN==1){} //RCEN cleared automatically when SSP1IF goes high

        data[i]=I2C2RCV;

        //ACK sequence
        if (i<N-1){ I2C2CONbits.ACKDT=0; } //jedes byte mit ACK bestätigen
        else {I2C2CONbits.ACKDT=1;} //send NACK if this is the last Byte

        I2C2CONbits.ACKEN=1; //start ack/nack sequence
        while(I2C2CONbits.ACKEN==1){}

    } //end for loop
    return FSM2_Stop;
}

void *FSM2_Stop(void)
{
    I2C2CONbits.PEN=1;
    while(I2C2CONbits.PEN==1){} //wait for the stop interrupt

    float temp = data[0]«8|data[1];
    char str[16];
    sprintf(str,"%f",temp/256);
    putsUART("Temperatur: ");
    putsUART(str);
    putsUART("°C");
    putsUART("\n");

    return FSM2_Idle;
}



/* Main Program                                                               */


int16_t main(void)
{
    DELAY_ANPASSUNG = ((SYS_FREQ/96)*2180ull)/1000000ull; //Berechnung der Delay Anpassung
    //uint16_t Count = 0;
    /* Configure the oscillator for the device */
    ConfigureOscillator();
    /* Initialize IO ports and peripherals */
    InitApp();

    //initUART();
    //init_timer1();
    init_ms_t4();
    //initI2C();


    TRISBbits.TRISB8 = 0; //LED0 als Ausgang
    ANSELBbits.ANSB8 = 0; //LED0 als Digitaler Ausgang

    TRISBbits.TRISB9 = 0; //LED als Ausgang
    ANSELBbits.ANSB9 = 0;

   //Taster als Eingänge
    _TRISG12 = 1;
    //Pull-up Widerstände einschalten
    _CNPUG12 = 1;


    _RP66R = _RPOUT_U1TX; //UART Pin Mapping
    RPINR18bits.U1RXR = 0b1011000;
    /* TODO <INSERT USER APPLICATION CODE HERE> */

    while(1)
    {
        PORTBbits.RB8=1;
        delay_ms(200);
        PORTBbits.RB8=0;
        delay_ms(200);
        //if(_T4IF)
        //{
            //_T4IF=0;
            //Count++;
            //if (Count >= HEARTBEAT_MS)
            //{
                //Count = 0;
                //Temp_FSM2();
            //}

        //}
    }
}
```

5.11 system.c

```

/* Files to Include                                                           */

/* Device header file */
#if defined(__XC16__)
    #include <xc.h>
#elif defined(__C30__)
    #if defined(__dsPIC33E__)
        #include <p33Exxxx.h>
    #elif defined(__dsPIC33F__)
        #include <p33Fxxxx.h>
    #endif
#endif


#include <stdint.h>          /* For uint16_t definition                       */
#include <stdbool.h>         /* For true/false definition                     */

```

```
#include "system.h"          /* variables/params used by system.c             */


/* System Level Functions                                                     */
/*                                                                            */
/* Custom oscillator configuration funtions, reset source evaluation          */
/* functions, and other non-peripheral microcontroller initialization         */
/* functions get placed in system.c.                                          */
/*                                                                            */

/* Refer to the device Family Reference Manual Oscillator section for
information about available oscillator configurations.  Typically
this would involve configuring the oscillator tuning register or clock
switching useing the compiler's __builtin_write_OSCCON functions.
Refer to the C Compiler for PIC24 MCUs and dsPIC DSCs User Guide in the
compiler installation directory /doc folder for documentation on the
__builtin functions.*/

/* TODO Add clock switching code if appropriate.  An example stub is below.   */
void ConfigureOscillator(void)
{
    if (SYS_FREQ>7370000L){//Nur umschalten auf Primary wenn höhere Frequenz erwünscht
        switch (SYS_FREQ){
            case 8000000L:
                //PLL muss nicht konfiguriert werden
                // externer Quartz mit 8Mhz
                break;
            case 50000000L:
                CLKDIVbits.PLLPOST=2; //N2=4
                PLLFBD=48; //M=50
                CLKDIVbits.PLLPRE=0; //N1=2
                break;
            case 70000000L:
                CLKDIVbits.PLLPOST=2; //N2=4
                PLLFBD=188; //M=190
                CLKDIVbits.PLLPRE=3; //N1=5
                break;
            case 100000000L:
                CLKDIVbits.PLLPOST=0; //N2=2
                PLLFBD=123; //M=125
                CLKDIVbits.PLLPRE=3; //N1=5
                break;
            case 140000000L:
                CLKDIVbits.PLLPOST=0; //N2=2
                PLLFBD=173; //M=175
                CLKDIVbits.PLLPRE=3; //N1=5
                break;
            //default:
                //#error Tets
        }
        OSCTUN = 0;

        if (SYS_FREQ == 8000000L){
            __builtin_write_OSCCONH(0x02); //Switch auf Primary ohne PLL

            __builtin_write_OSCCONL(OSCCON | 0x01);
            while (OSCCONbits.COSC!= 0x02); //Warten bis gewechselt wurde
            }

        else {
            __builtin_write_OSCCONH(0x03); //Switch auf Primary mit PLL

            __builtin_write_OSCCONL(OSCCON | 0x01);

            while (OSCCONbits.COSC!= 0x3); //Warten bis gewechselt wurde
            while (OSCCONbits.LOCK!= 1);
        }

    }
}


//Timer1
void init_timer1(){
    __builtin_write_OSCCONL(0b00000011); //SOSC aktivieren
    T1CONbits.TON = 0; // Disable Timer
    T1CONbits.TCS = 1; // Select external clock
    T1CONbits.TSYNC = 0; // Disable Synchronization
    T1CONbits.TCKPS = 0b00; // Select 1:1 Prescaler
    TMR1 = 0x00; // Clear timer register
    PR1 = 32767; // Load the period value, Quarztakt

    IPC0bits.T1IP = 2; // Set Timer 1 Interrupt Priority Level
    IFS0bits.T1IF = 0; // Clear Timer 1 Interrupt Flag
    IEC0bits.T1IE = 1; // Enable Timer1 interrupt
    T1CONbits.TON = 1; // Start Timer
}
```

```

void init_ms_t4(){
    T4CONbits.TON = 0; // Stop any 16/32-bit Timer3 operation
    T4CONbits.TCS = 0; // Select internal instruction cycle clock

    T4CONbits.TGATE = 0; // Disable Gated Timer mode
    T4CONbits.TCKPS = 0b10; // Select 1:64 Prescaler
    TMR4 = 0x00; // Clear
    PR4 = (FCY/64000)-1; // Load 32-bit period value (lsw)
    //IFS0bits.T2IF = 0; // Clear Timer2 Interrupt Flag
    //IEC0bits.T2IE = 0; // Disable Timer2 interrupt
    T4CONbits.TON = 1; // Start 32-bit Timer
}
```

5.12 system.h

```

/* System Level #define Macros                                                */

/* TODO Define system operating frequency */

/* Microcontroller MIPs (FCY) */
//#define SYS_FREQ        7370000L
//#define SYS_FREQ        8000000L
#define SYS_FREQ        50000000L
//#define SYS_FREQ        70000000L
//#define SYS_FREQ        100000000L
//#define SYS_FREQ        140000000L

#define FCY             SYS_FREQ/2



/* System Function Prototypes                                                 */

/* Custom oscillator configuration funtions, reset source evaluation
functions, and other non-peripheral microcontroller initialization functions
go here. */


//System Prototypen
void ConfigureOscillator(void); /* Handles clock switching/osc initialization */

void init_timer1(void);
void init_ms_t4(void);

```

5.13 traps.c

```

/* Files to Include                                                           */

/* Device header file */
#if defined(__XC16__)
    #include <xc.h>
#elif defined(__C30__)
    #if defined(__dsPIC33E__)
        #include <p33Exxxx.h>
    #elif defined(__dsPIC33F__)
        #include <p33Fxxxx.h>
    #endif
#endif

#include <stdint.h>        /* Includes uint16_t definition */
#include <stdbool.h>       /* Includes true/false definition */


/* Trap Function Prototypes                                                   */

/* <Other function prototypes for debugging trap code may be inserted here>   */

/* Use if INTCON2 ALTIVT=1 */
void __attribute__((interrupt,no_auto_psv)) _OscillatorFail(void);
void __attribute__((interrupt,no_auto_psv)) _AddressError(void);
void __attribute__((interrupt,no_auto_psv)) _StackError(void);
void __attribute__((interrupt,no_auto_psv)) _MathError(void);

#if defined(__HAS_DMA__)

```

```
void __attribute__((interrupt,no_auto_psv)) _DMACError(void);

#endif

#if defined(__dsPIC33F__)

/* Use if INTCON2 ALTIVT=0 */
void __attribute__((interrupt,no_auto_psv)) _AltOscillatorFail(void);
void __attribute__((interrupt,no_auto_psv)) _AltAddressError(void);
void __attribute__((interrupt,no_auto_psv)) _AltStackError(void);
void __attribute__((interrupt,no_auto_psv)) _AltMathError(void);

    #if defined(__HAS_DMA__)

    void __attribute__((interrupt,no_auto_psv)) _AltDMACError(void);

    #endif

#endif

/* Default interrupt handler */
void __attribute__((interrupt,no_auto_psv)) _DefaultInterrupt(void);

#if defined(__dsPIC33E__)

/* These are additional traps in the 33E family.  Refer to the PIC33E
migration guide.  There are no Alternate Vectors in the 33E family. */
void __attribute__((interrupt,no_auto_psv)) _HardTrapError(void);
void __attribute__((interrupt,no_auto_psv)) _SoftTrapError(void);

#endif


/* Trap Handling                                                               */
/*                                                                             */
/* These trap routines simply ensure that the device continuously loops        */
/* within each routine.  Users who actually experience one of these traps      */
/* can add code to handle the error.  Some basic examples for trap code,       */
/* including assembly routines that process trap sources, are available at     */
/* www.microchip.com/codeexamples                                              */

/* Primary (non-alternate) address error trap function declarations */
void __attribute__((interrupt,no_auto_psv)) _OscillatorFail(void)
{
        INTCON1bits.OSCFAIL = 0;         /* Clear the trap flag */
        while(1);
}

void __attribute__((interrupt,no_auto_psv)) _AddressError(void)
{
        INTCON1bits.ADDRERR = 0;         /* Clear the trap flag */
        while (1);
}
void __attribute__((interrupt,no_auto_psv)) _StackError(void)
{
        INTCON1bits.STKERR = 0;          /* Clear the trap flag */
        while (1);
}

void __attribute__((interrupt,no_auto_psv)) _MathError(void)
{
        INTCON1bits.MATHERR = 0;         /* Clear the trap flag */
        while (1);
}

#if defined(__HAS_DMA__)

void __attribute__((interrupt,no_auto_psv)) _DMACError(void)
{
        INTCON1bits.DMACERR = 0;         /* Clear the trap flag */
        while (1);
}

#endif

#if defined(__dsPIC33F__)

/* Alternate address error trap function declarations */
void __attribute__((interrupt,no_auto_psv)) _AltOscillatorFail(void)
{
        INTCON1bits.OSCFAIL = 0;         /* Clear the trap flag */
        while (1);
}

void __attribute__((interrupt,no_auto_psv)) _AltAddressError(void)
{
        INTCON1bits.ADDRERR = 0;         /* Clear the trap flag */
        while (1);
}

void __attribute__((interrupt,no_auto_psv)) _AltStackError(void)
{
        INTCON1bits.STKERR = 0;         /* Clear the trap flag */
        while (1);
}

void __attribute__((interrupt,no_auto_psv)) _AltMathError(void)
{
        INTCON1bits.MATHERR = 0;        /* Clear the trap flag */
        while (1);
}

    #if defined(__HAS_DMA__)

    void __attribute__((interrupt,no_auto_psv)) _AltDMACError(void)
    {
         INTCON1bits.DMACERR = 0;        /* Clear the trap flag */
         while (1);
    }

    #endif

#endif


/* Default Interrupt Handler                                                  */
/*                                                                            */
/* This executes when an interrupt occurs for an interrupt source with an    */
/* improperly defined or undefined interrupt handling routine.               */

void __attribute__((interrupt,no_auto_psv)) _DefaultInterrupt(void)
{
        while(1);
}

#if defined(__dsPIC33E__)

/* These traps are new to the dsPIC33E family.  Refer to the device Interrupt
chapter of the FRM to understand trap priority. */
void __attribute__((interrupt,no_auto_psv)) _HardTrapError(void)
{
    while(1);
}
void __attribute__((interrupt,no_auto_psv)) _SoftTrapError(void)
{
    while(1);
}

#endif
```

5.14 user.c

```

/* Files to Include                                                           */

/* Device header file */
#if defined(__XC16__)
    #include <xc.h>
#elif defined(__C30__)
    #if defined(__dsPIC33E__)
        #include <p33Exxxx.h>
    #elif defined(__dsPIC33F__)
        #include <p33Fxxxx.h>
    #endif
#endif

#include <stdint.h>          /* For uint16_t definition                       */
#include <stdbool.h>         /* For true/false definition                     */
#include "user.h"            /* variables/params used by user.c               */


/* User Functions                                                             */

/* <Initialize variables in user.h and insert code for user algorithms.> */

void InitApp(void)
{
    /* TODO Initialize User Ports/Peripherals/Project here */

    /* Setup analog functionality and port direction */

    /* Initialize peripherals */
}

void setLED(uint16_t nr)
{
   if (nr>=4) return;
   LATB = LATB | (1 « nr+8);
}



//4-bit Wort -> RB8-11

//uint16_t leds=0b 0000 0000 0000 1101;

//LATB = (LATB & ~0b0000111100000000) | ((leds«8) &0b0000111100000000);
```

5.15 user.h

```

/* User Level #define Macros                                                  */

#define LED0 _LATB8
#define LED1 _LATB9
#define LED2 _LATB10
#define LED3 _LATB11
#define T0 !_RG12
#define T1 !_RG13
#define T2 !_RG14
#define T3 !_RG15
/* TODO Application specific user parameters used in user.c may go here */


/* User Function Prototypes                                                   */

/* TODO User level functions prototypes (i.e. InitApp) go here */

void InitApp(void);         /* I/O and Peripheral Initialization */

void setLED(uint16_t nr);
```

Index