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

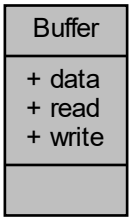
Hier folgt die Aufzählung aller dokumentierten Dateien mit einer Kurzbeschreibung:

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

```
00001 build/XC16_dsPIC33EP512MU810/production/configuration_bits.o: \
00002  configuration_bits.c
```

5.2 interrupts.o.d

```
00001 build/XC16_dsPIC33EP512MU810/production/interrupts.o: interrupts.c
```

5.3 main.o.d

```
00001 build/XC16_dsPIC33EP512MU810/production/main.o: main.c system.h user.h
00002
00003 system.h:
00004
00005 user.h:
```

5.4 system.o.d

```
00001 build/XC16_dsPIC33EP512MU810/production/system.o: system.c system.h
00002
00003 system.h:
```

5.5 traps.o.d

```
00001 build/XC16_dsPIC33EP512MU810/production/traps.o: traps.c
```

5.6 user.o.d

```
00001 build/XC16_dsPIC33EP512MU810/production/user.o: user.c user.h
00002
00003 user.h:
```

5.7 configuration_bits.c

```
00001
00002 // DSPIC33EP512MU810 Configuration Bit Settings
00003
00004 // 'C' source line config statements
00005
00006 // FGS
00007 #pragma config GWRP = OFF               // General Segment Write-Protect bit (General Segment may be
       written)
00008 #pragma config GSS = OFF                // General Segment Code-Protect bit (General Segment Code
       protect is disabled)
00009 #pragma config GSSK = OFF               // General Segment Key bits (General Segment Write Protection
       and Code Protection is Disabled)
00010
00011 // FOSCSEL
00012 #pragma config FNOSC = FRCDIVN          // Initial Oscillator Source Selection Bits (Internal Fast RC
       (FRC) Oscillator with postscaler)
00013 #pragma config IESO = ON                // Two-speed Oscillator Start-up Enable bit (Start up device
       with FRC, then switch to user-selected oscillator source)
00014
00015 // FOSC
00016 #pragma config POSCMD = XT              // Primary Oscillator Mode Select bits (XT Crystal Oscillator
       Mode)
00017 #pragma config OSCIOFNC = OFF           // OSC2 Pin Function bit (OSC2 is clock output)
00018 #pragma config IOL1WAY = ON             // Peripheral pin select configuration (Allow only one
       reconfiguration)
00019 #pragma config FCKSM = CSECMD           // Clock Switching Mode bits (Clock switching is
       enabled,Fail-safe Clock Monitor is disabled)
00020
00021 // FWDT
00022 #pragma config WDTPOST = PS32768        // Watchdog Timer Postscaler Bits (1:32,768)
00023 #pragma config WDTPRE = PR128           // Watchdog Timer Prescaler bit (1:128)
00024 #pragma config PLLKEN = ON              // PLL Lock Wait Enable bit (Clock switch to PLL source will
       wait until the PLL lock signal is valid.)
00025 #pragma config WINDIS = OFF             // Watchdog Timer Window Enable bit (Watchdog Timer in
       Non-Window mode)
00026 #pragma config FWDTEN = ON              // Watchdog Timer Enable bit (Watchdog timer always enabled)
00027
00028 // FPOR
00029 #pragma config FPWRT = PWR128           // Power-on Reset Timer Value Select bits (128ms)
00030 #pragma config BOREN = ON               // Brown-out Reset (BOR) Detection Enable bit (BOR is enabled)
00031 #pragma config ALTI2C1 = OFF            // Alternate I2C pins for I2C1 (SDA1/SCK1 pins are selected as
       the I/O pins for I2C1)
00032 #pragma config ALTI2C2 = ON            // Alternate I2C pins for I2C2 (SDA2/SCK2 pins are selected as
       the I/O pins for I2C2)
00033
00034 // FICD
00035 #pragma config ICS = PGD1               // ICD Communication Channel Select bits (Communicate on PGEC1
       and PGED1)
00036 #pragma config RSTPRI = PF              // Reset Target Vector Select bit (Device will obtain reset
       instruction from Primary flash)
00037 #pragma config JTAGEN = OFF             // JTAG Enable bit (JTAG is disabled)
00038
00039 // FAS
00040 #pragma config AWRP = OFF               // Auxiliary Segment Write-protect bit (Auxiliary program
       memory is not write-protected)
00041 #pragma config APL = OFF                // Auxiliary Segment Code-protect bit (Aux Flash Code protect
       is disabled)
00042 #pragma config APLK = OFF               // Auxiliary Segment Key bits (Aux Flash Write Protection and
       Code Protection is Disabled)
00043
00044 // #pragma config statements should precede project file includes.
00045 // Use project enums instead of #define for ON and OFF.
00046
00047 #include <xc.h>
00048
00049
00050
00051
```

5.8 interrupts.c

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

```
00012         #include <p33Fxxxx.h>
00013     #endif
00014 #endif
00015
00016 #include <stdint.h>        /* Includes uint16_t definition   */
00017 #include <stdbool.h>       /* Includes true/false definition */
00018
00019
00020 /* Interrupt Vector Options                                                */
00021
00022 /*                                                                         */
00023 /* Refer to the C30 (MPLAB C Compiler for PIC24F MCUs and dsPIC33F DSCs) User */
00024 /* Guide for an up to date list of the available interrupt options.        */
00025 /* Alternately these names can be pulled from the device linker scripts.   */
00026 /*                                                                         */
00027 /* dsPIC33F Primary Interrupt Vector Names:                                */
00028 /*                                                                         */
00029 /* _INT0Interrupt      _C1Interrupt                                        */
00030 /* _IC1Interrupt       _DMA3Interrupt                                      */
00031 /* _OC1Interrupt       _IC3Interrupt                                       */
00032 /* _T1Interrupt        _IC4Interrupt                                       */
00033 /* _DMA0Interrupt      _IC5Interrupt                                       */
00034 /* _IC2Interrupt       _IC6Interrupt                                       */
00035 /* _OC2Interrupt       _OC5Interrupt                                       */
00036 /* _T2Interrupt        _OC6Interrupt                                       */
00037 /* _T3Interrupt        _OC7Interrupt                                       */
00038 /* _SPI1ErrInterrupt   _OC8Interrupt                                       */
00039 /* _SPI1Interrupt      _DMA4Interrupt                                      */
00040 /* _U1RXInterrupt      _T6Interrupt                                        */
00041 /* _U1TXInterrupt      _T7Interrupt                                        */
00042 /* _ADC1Interrupt      _SI2C2Interrupt                                     */
00043 /* _DMA1Interrupt      _MI2C2Interrupt                                     */
00044 /* _SI2C1Interrupt     _T8Interrupt                                        */
00045 /* _MI2C1Interrupt     _T9Interrupt                                        */
00046 /* _CNInterrupt        _INT3Interrupt                                      */
00047 /* _INT1Interrupt      _INT4Interrupt                                      */
00048 /* _ADC2Interrupt      _C2RxRdyInterrupt                                   */
00049 /* _DMA2Interrupt      _C2Interrupt                                        */
00050 /* _OC3Interrupt       _DCIErrInterrupt                                    */
00051 /* _OC4Interrupt       _DCIInterrupt                                       */
00052 /* _T4Interrupt        _DMA5Interrupt                                      */
00053 /* _T5Interrupt        _U1ErrInterrupt                                     */
00054 /* _INT2Interrupt      _U2ErrInterrupt                                     */
00055 /* _U2RXInterrupt      _DMA6Interrupt                                      */
00056 /* _U2TXInterrupt      _DMA7Interrupt                                      */
00057 /* _SPI2ErrInterrupt   _C1TxReqInterrupt                                   */
00058 /* _SPI2Interrupt      _C2TxReqInterrupt                                   */
00059 /* _C1RxRdyInterrupt                                                       */
00060 /*                                                                         */
00061 /* dsPIC33E Primary Interrupt Vector Names:                                */
00062 /*                                                                         */
00063 /* _INT0Interrupt      _IC4Interrupt      _U4TXInterrupt                   */
00064 /* _IC1Interrupt       _IC5Interrupt      _SPI3ErrInterrupt                */
00065 /* _OC1Interrupt       _IC6Interrupt      _SPI3Interrupt                   */
00066 /* _T1Interrupt        _OC5Interrupt      _OC9Interrupt                    */
00067 /* _DMA0Interrupt      _OC6Interrupt      _IC9Interrupt                    */
00068 /* _IC2Interrupt       _OC7Interrupt      _PWM1Interrupt                   */
00069 /* _OC2Interrupt       _OC8Interrupt      _PWM2Interrupt                   */
00070 /* _T2Interrupt        _PMPInterrupt      _PWM3Interrupt                   */
00071 /* _T3Interrupt        _DMA4Interrupt     _PWM4Interrupt                   */
00072 /* _SPI1ErrInterrupt   _T6Interrupt       _PWM5Interrupt                   */
00073 /* _SPI1Interrupt      _T7Interrupt       _PWM6Interrupt                   */
00074 /* _U1RXInterrupt      _SI2C2Interrupt    _PWM7Interrupt                   */
00075 /* _U1TXInterrupt      _MI2C2Interrupt    _DMA8Interrupt                   */
00076 /* _AD1Interrupt       _T8Interrupt       _DMA9Interrupt                   */
00077 /* _DMA1Interrupt      _T9Interrupt       _DMA10Interrupt                  */
00078 /* _NVMInterrupt       _INT3Interrupt     _DMA11Interrupt                  */
00079 /* _SI2C1Interrupt     _INT4Interrupt     _SPI4ErrInterrupt                */
00080 /* _MI2C1Interrupt     _C2RxRdyInterrupt  _SPI4Interrupt                   */
00081 /* _CM1Interrupt       _C2Interrupt       _OC10Interrupt                   */
00082 /* _CNInterrupt        _QEI1Interrupt     _IC10Interrupt                   */
00083 /* _INT1Interrupt      _DCIEInterrupt     _OC11Interrupt                   */
00084 /* _AD2Interrupt       _DCIInterrupt      _IC11Interrupt                   */
00085 /* _IC7Interrupt       _DMA5Interrupt     _OC12Interrupt                   */
00086 /* _IC8Interrupt       _RTCCInterrupt     _IC12Interrupt                   */
00087 /* _DMA2Interrupt      _U1ErrInterrupt    _DMA12Interrupt                  */
00088 /* _OC3Interrupt       _U2ErrInterrupt    _DMA13Interrupt                  */
00089 /* _OC4Interrupt       _CRCInterrupt      _DMA14Interrupt                  */
00090 /* _T4Interrupt        _DMA6Interrupt     _OC13Interrupt                   */
00091 /* _T5Interrupt        _DMA7Interrupt     _IC13Interrupt                   */
00092 /* _INT2Interrupt      _C1TxReqInterrupt  _OC14Interrupt                   */
00093 /* _U2RXInterrupt      _C2TxReqInterrupt  _IC14Interrupt                   */
00094 /* _U2TXInterrupt      _QEI2Interrupt     _OC15Interrupt                   */
00095 /* _SPI2ErrInterrupt   _U3ErrInterrupt    _IC15Interrupt                   */
00096 /* _SPI2Interrupt      _U3RXInterrupt     _OC16Interrupt                   */
00097 /* _C1RxRdyInterrupt   _U3TXInterrupt     _IC16Interrupt                   */
00098 /* _C1Interrupt        _USB1Interrupt     _ICDInterrupt                    */
```

```
00099 /* _DMA3Interrupt     _U4ErrInterrupt    _PWMSpEventMatchInterrupt           */
00100 /* _IC3Interrupt      _U4RXInterrupt     _PWMSecSpEventMatchInterrupt        */
00101 /*                                                                          */
00102 /* For alternate interrupt vector naming, simply add 'Alt' between the prim.  */
00103 /* interrupt vector name '_' and the first character of the primary interrupt */
00104 /* vector name.  There is no Alternate Vector or 'AIVT' for the 33E family.   */
00105 /*                                                                          */
00106 /* For example, the vector name _ADC2Interrupt becomes _AltADC2Interrupt in   */
00107 /* the alternate vector table.                                              */
00108 /*                                                                          */
00109 /* Example Syntax:                                                          */
00110 /*                                                                          */
00111 /* void __attribute__((interrupt,auto_psv)) <Vector Name>(void)             */
00112 /* {                                                                        */
00113 /*     <Clear Interrupt Flag>                                               */
00114 /* }                                                                        */
00115 /*                                                                          */
00116 /* For more comprehensive interrupt examples refer to the C30 (MPLAB C      */
00117 /* Compiler for PIC24 MCUs and dsPIC DSCs) User Guide in the                 */
00118 /* <C30 compiler instal directory>/doc directory for the latest compiler    */
00119 /* release.  For XC16, refer to the MPLAB XC16 C Compiler User's Guide in the */
00120 /* <XC16 compiler instal directory>/doc folder.                             */
00121 /*                                                                          */
00122
00123 /* Interrupt Routines                                                       */
00124
00126 /* TODO Add interrupt routine code here. */
```

5.9 main.c

```
00001
00002 /* Files to Include                                                       */
00003
00004 #include <xc.h>
00005
00006 #include <stdint.h>        /* Includes uint16_t definition                */
00007 #include <stdbool.h>       /* Includes true/false definition              */
00008 #include <string.h>
00009 #include <stdio.h>
00010 #include <stdlib.h>
00011
00012 #include "system.h"        /* System funct/params, like osc/peripheral config */
00013 #include "user.h"          /* User funct/params, such as InitApp          */
00014
00015
00016 /* Global Variable Declaration                                           */
00017
00018 #define HEARTBEAT_MS 1
00019 uint32_t DELAY_ANPASSUNG;
00020
00021 //UART
00022 #define BAUDRATE 9600
00023 #define BRGVAL ((FCY/BAUDRATE)/16)-1
00024
00025 //FIFO
00026 #define BUFFER_FAIL     0
00027 #define BUFFER_SUCCESS  1
00028 #define BUFFER_SIZE 128
00029
00030
00031 //I2C
00032 #define I2C_SCL      _RA2
00033 #define I2C_SDA      _RA3
00034 #define I2C_SCL_TRIS _TRISA2
00035 #define I2C_SDA_TRIS _TRISA3
00036 uint8_t data[2];
00037
00038 /*Typen-Definitionen*****************************************************/
00039 typedef struct {
00040   uint8_t data[BUFFER_SIZE];
00041   uint8_t read; // zeigt auf das Feld mit dem ältesten Inhalt
00042   uint8_t write; // zeigt immer auf leeres Feld
00043 }Buffer;
00044
00045 Buffer FIFO = {{}, 0, 0};
00046
00047 typedef void *(*StateFunc)();
00048
00049
00050 /*Prototypes*************************************************************/
00051
00052
00053 int16_t putsUART(const char *str);
```

```
00054 int16_t getcFIFO_TX(volatile uint16_t *c);
00055 //int16_t getcFIFO_RX(char *c);
00056
00057 int16_t putcFIFO_TX(char c);
00058 //int16_t putcFIFO_RX(char c);
00059
00060 void *FSM2_Idle(void);
00061 void *FSM2_Start(void);
00062 void *FSM2_Adresse(void);
00063 void *FSM2_ACK_Receive(void);
00064 void *FSM2_Data_Receive(void);
00065 void *FSM2_Stop(void);
00066 void Temp_FSM2(void);
00067
00068 /*Funktionen*****************************************************************/
00069
00074 void delay_ms(uint16_t milliseconds) {
00075     uint32_t i=0;
00076     for (i=0;i<(DELAY_ANPASSUNG*(uint32_t)milliseconds);i++){
00077          }
00078 }
00079
00080 void __attribute__((__interrupt__, no_auto_psv)) _T1Interrupt(void)
00081     {
00082         _T1IF = 0; //Clear Timer1 interrupt flag
00083
00084         putsUART("Hello World\n");
00085
00086     }
00087
00088 //UART
00089 void initUART(){
00090     U1MODEbits.STSEL = 0; // 1-Stop bit
00091     U1MODEbits.PDSEL = 0; // No Parity, 8-Data bits
00092     U1MODEbits.ABAUD = 0; // Auto-Baud disabled
00093     U1MODEbits.UEN = 0;
00094     U1MODEbits.LPBACK = 0;
00095     U1MODEbits.RXINV = 0;
00096     //U1MODEbits.ALTIO = 0;
00097
00098     U1MODEbits.URXINV = 0;
00099     U1MODEbits.RTSMD = 0;
00100
00101     U1MODEbits.BRGH = 0; // Standard-Speed mode
00102     U1BRG = BRGVAL; // Baud Rate setting for 9600
00103
00104     U1STAbits.UTXISEL0 = 0; // Interrupt after one TX character is transmitted
00105     U1STAbits.UTXISEL1 = 0;
00106     U1STAbits.UTXBRK = 0;
00107     U1STAbits.ADDEN = 0;
00108     U1STAbits.UTXINV = 0;
00109     U1STAbits.URXISEL = 0;
00110     U1STA = U1STA | 0b0001000000000000;
00111     //_URXEN = 1;
00112
00113     //_U1RXIE = 1; // Enable UART RX interrupt
00114
00115     U1MODEbits.UARTEN = 1; // Enable UART
00116     //delay_ms(2);
00117     U1STAbits.UTXEN = 1; // Enable UART TX
00118
00119     /* Wait at least 105 microseconds (1/9600) before sending first char */
00120     delay_ms(2);
00121     _U1TXIE = 1; // Enable UART TX interrupt
00122
00123 }
00124
00125 void __attribute__((__interrupt__, no_auto_psv)) _U1TXInterrupt(void)
00126 {
00127     _U1TXIF = 0; // Clear TX Interrupt flag
00128
00129     getcFIFO_TX(&U1TXREG);
00130
00131 }
00132
00133
00134
00135
00136 int16_t putcFIFO_TX(char c)
00137 {
00138   //if (buffer.write >= BUFFER_SIZE)
00139   //  buffer.write = 0; // erhöht sicherheit
00140   _LATF0 = 1;
00141   if ( ( FIFO.write + 1 == FIFO.read ) ||
00142        ( FIFO.read == 0 && FIFO.write + 1 == BUFFER_SIZE ) )
00143     return BUFFER_FAIL; // voll
00144
```

```
00145   FIFO.data[FIFO.write] = c;
00146
00147   FIFO.write++;
00148   if (FIFO.write >= BUFFER_SIZE)
00149     FIFO.write = 0;
00150
00151   return BUFFER_SUCCESS;
00152 }
00153
00154 int16_t getcFIFO_TX(volatile uint16_t *c)
00155 {
00156   _LATF0 = 1;
00157   if (FIFO.read == FIFO.write)
00158     return BUFFER_FAIL;
00159
00160   *c = FIFO.data[FIFO.read];
00161
00162   FIFO.read++;
00163   if (FIFO.read >= BUFFER_SIZE)
00164     FIFO.read = 0;
00165
00166   return BUFFER_SUCCESS;
00167 }
00168
00169 int16_t putcUART(char c){
00170     _LATF0 = 1;
00171     _GIE = 0; // Interrupts ausschalten
00172     int16_t erfolg = putcFIFO_TX(c);
00173     _GIE = 1;
00174     return erfolg;
00175
00176
00177 }
00178
00179 int16_t putsUART(const char *str) {
00180     _LATF0 = 1;
00181     uint16_t i;
00182     uint16_t length = strlen(str);
00183
00184     _GIE = 0;   //Global Interrupt disable
00185     for(i = 0; i < length; i++) {
00186         //uint16_t ret = putcFIFO_TX(str[i]);
00187         if(! putcFIFO_TX(str[i]))
00188             break;
00189     }
00190     _GIE = 1;
00191     int16_t erfolg = -i;
00192     if(erfolg == -length)
00193         erfolg *= -1;
00194     _U1TXIF = 1; //Interuppt Routine Starten um FIFO-Inhalt zu senden
00195     return erfolg;
00196 }
00197
00198 //I2C
00199
00200 void Temp_FSM2(void)
00201 {
00202      static StateFunc statefunc = FSM2_Idle;
00203
00204      statefunc = (StateFunc)(*statefunc)();
00205 }
00206
00207 void initI2C(){
00208     I2C2CONbits.A10M = 0;
00209     I2C2BRG = 245; //100kHz
00210
00211     // Einschalten I2C mit eigenem Workaround, I2C Peripheriemodul kann hier leider nicht verwendet
       werden
00212     I2C_SDA_TRIS = 1;    // Pins wie einen Open-Kollektor-Treiber verwenden, d.h. 1 - hochohmig, 0
       wird getrieben
00213     I2C_SCL_TRIS = 1;
00214     I2C_SDA = 0;
00215     I2C_SCL = 0;
00216
00217         int j;
00218         for (j=0; j<=9; j++)    // takten bis min 1 Byte
00219         {
00220           I2C_SCL_TRIS = 0; delay_ms(1); // 5 us wÃ¤ren ausreichend ...100 kBaud
00221           I2C_SCL_TRIS = 1; delay_ms(1);
00222         }
00223         // Start Condition senden
00224         I2C_SCL_TRIS = 0;  delay_ms(1);
00225         I2C_SDA_TRIS = 0;  delay_ms(1);
00226         // Stop Condition senden
00227         I2C_SCL_TRIS = 1;  delay_ms(1);
00228         I2C_SDA_TRIS = 1;  delay_ms(1);
00229
```

```
00230         // Nun I2C erst anschalten
00231         _MI2C2IF = 0; //Interrupt falls noetig
00232         _MI2C2IE = 0;
00233         I2C2CONbits.I2CEN = 1;
00234 
00235     //Sensor Pointer auf TEMP Register setzten
00236     I2C2CONbits.SEN=1; //start
00237     while(I2C2CONbits.SEN==1){}
00238 
00239     //Tx Device address + Write bit
00240     I2C2TRN=0b10010000;
00241     while(I2C2STATbits.TRSTAT==1){}
00242 
00243     if (I2C2STATbits.ACKSTAT==1){   //if NACK received, generate stop condition and exit
00244         I2C2STATbits.ACKSTAT=0;
00245         I2C2CONbits.PEN=1;
00246         while(I2C2CONbits.PEN==1){} //wait for the stop interrupt;
00247         return;
00248     }
00249 
00250     //Tx Register Address
00251     I2C2TRN=0b00000000; //Pointer auf TEMP REGISTER setzten
00252     while(I2C2STATbits.TRSTAT==1){}
00253 
00254     if (I2C2STATbits.ACKSTAT==1){   //if NACK received, generate stop condition and exit
00255         I2C2STATbits.ACKSTAT=0;
00256         I2C2CONbits.PEN=1;
00257         while(I2C2CONbits.PEN==1){} //wait for the stop interrupt;
00258         return;
00259     }
00260 
00261     I2C2CONbits.PEN=1; //stop
00262     while(I2C2CONbits.PEN==1){} //wait for the stop interrupt
00263 }
00264 
00265 
00266 
00267 void *FSM2_Idle(void)
00268 {
00269     static int c = 0;
00270     if (c>=999){
00271         c=0;
00272         return FSM2_Start;
00273     }
00274     c++;
00275     return FSM2_Idle;
00276 
00277 }
00278 
00279 void *FSM2_Start(void)
00280 {
00281     I2C2CONbits.SEN=1; //Start
00282     while(I2C2CONbits.SEN==1){}
00283     return FSM2_Adresse;
00284 }
00285 
00286 void *FSM2_Adresse(void)
00287 {
00288     //Tx Device address + Read bit
00289     I2C2TRN=0b10010001;
00290     while(I2C2STATbits.TRSTAT==1){} //Warten solange übertragen wird
00291     return FSM2_ACK_Receive;
00292 }
00293 
00294 void *FSM2_ACK_Receive(void)
00295 {
00296     if (I2C2STATbits.ACKSTAT==1){   //if NACK received, generate stop condition and exit
00297         I2C2STATbits.ACKSTAT=0;
00298         return FSM2_Stop;
00299     }
00300     return FSM2_Data_Receive;
00301 }
00302 
00303 void *FSM2_Data_Receive(void)
00304 {
00305     int N=2; //2 bytes empfangen
00306     int i;
00307 
00308     for(i=0;i<N;i++){
00309         I2C2CONbits.RCEN=1; //Empfangen aktivieren
00310         while(I2C2CONbits.RCEN==1){} //RCEN cleared automatically when SSP1IF goes high
00311 
00312         data[i]=I2C2RCV;
00313 
00314         //ACK sequence
00315         if (i<N-1){ I2C2CONbits.ACKDT=0; } //jedes byte mit ACK bestätigen
00316         else {I2C2CONbits.ACKDT=1;} //send NACK if this is the last Byte
```

```
00317
00318         I2C2CONbits.ACKEN=1; //start ack/nack sequence
00319         while(I2C2CONbits.ACKEN==1){}
00320
00321     } //end for loop
00322     return FSM2_Stop;
00323 }
00324
00325 void *FSM2_Stop(void)
00326 {
00327     I2C2CONbits.PEN=1;
00328     while(I2C2CONbits.PEN==1){} //wait for the stop interrupt
00329
00330     double temp = data[0]«8|data[1];
00331     char str[16];
00332     sprintf(str,"%f",temp/256);
00333     putsUART("Temperatur: ");
00334     putsUART(str);
00335     putsUART("°C");
00336     putsUART("\n");
00337
00338     return FSM2_Idle;
00339 }
00340
00341
00342
00343 /* Main Program                                                              */
00344
00346 int16_t main(void)
00347 {
00348     DELAY_ANPASSUNG = ((SYS_FREQ/96)*2180ull)/1000000ull; //Berechnung der Delay Anpassung
00349     uint16_t Count = 0;
00350     /* Configure the oscillator for the device */
00351     ConfigureOscillator();
00352     /* Initialize IO ports and peripherals */
00353     //InitApp();
00354     initUART();
00355     init_timer1();
00356     init_ms_t4();
00357     initI2C();
00358
00359
00360     TRISBbits.TRISB8 = 0; //LED als Ausgang
00361     ANSELBbits.ANSB8 = 0;
00362
00363     TRISBbits.TRISB9 = 0; //LED als Ausgang
00364     ANSELBbits.ANSB9 = 0;
00365
00366    //Taster als Eingänge
00367     _TRISG12 = 1;
00368     //Pull-up Widerstände einschalten
00369     _CNPUG12 = 1;
00370
00371
00372     _RP66R = _RPOUT_U1TX; //UART Pin Mapping
00373     RPINR18bits.U1RXR = 0b1011000;
00374
00375     while(1)
00376     {
00377         if(_T4IF)
00378         {
00379             _T4IF=0;
00380             Count++;
00381             if (Count >= HEARTBEAT_MS)
00382             {
00383                 Count = 0;
00384                 Temp_FSM2();
00385             }
00386
00387         }
00388     }
00389 }
```

5.10 main_less.c

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

```
00010         #include <p33Exxxx.h>
00011     #elif defined(__dsPIC33F__)
00012         #include <p33Fxxxx.h>
00013     #endif
00014 #endif
00015
00016
00017 #include <stdint.h>        /* Includes uint16_t definition                    */
00018 #include <stdbool.h>       /* Includes true/false definition                  */
00019 #include <string.h>
00020 #include <stdio.h>
00021 #include <stdlib.h>
00022
00023 #include "system.h"        /* System funct/params, like osc/peripheral config */
00024 #include "user.h"          /* User funct/params, such as InitApp              */
00025
00026
00027 /* Global Variable Declaration                                                */
00028
00029 #define HEARTBEAT_MS 1
00030 //UART
00031
00032 #define BAUDRATE 9600
00033 #define BRGVAL ((FCY/BAUDRATE)/16)-1
00034 //FIFO
00035
00036 #define BUFFER_FAIL     0
00037 #define BUFFER_SUCCESS  1
00038 #define BUFFER_SIZE 128
00039 uint32_t DELAY_ANPASSUNG;
00040
00041 //I2C
00042 #define I2C_SCL      _RA2
00043 #define I2C_SDA      _RA3
00044 #define I2C_SCL_TRIS _TRISA2
00045 #define I2C_SDA_TRIS _TRISA3
00046 uint8_t data[2];
00047
00048 /*Typen-Definitionen*****************************************************************/
00049
00050 typedef struct {
00051   uint8_t data[BUFFER_SIZE];
00052   uint8_t read; // zeigt auf das Feld mit dem ältesten Inhalt
00053   uint8_t write; // zeigt immer auf leeres Feld
00054 }Buffer;
00055
00056 Buffer FIFO = {{}, 0, 0};
00057
00058 typedef void *(*StateFunc)();
00059
00060
00061 /*Prototypes*************************************************************************/
00062 void init_ms_t4(void);
00063
00064 int16_t putsUART(const char *str);
00065 int16_t getcFIFO_TX(volatile uint16_t *c);
00066 //int16_t getcFIFO_RX(char *c);
00067
00068 int16_t putcFIFO_TX(char c);
00069 //int16_t putcFIFO_RX(char c);
00070
00071 void *FSM2_Idle(void);
00072 void *FSM2_Start(void);
00073 void *FSM2_Adresse(void);
00074 void *FSM2_ACK_Receive(void);
00075 void *FSM2_Data_Receive(void);
00076 void *FSM2_Stop(void);
00077 void Temp_FSM2(void);
00078
00079 /*Funktionen*************************************************************************/
00085 void delay_ms(uint16_t milliseconds) {
00086     uint32_t i=0;
00087     for (i=0;i<(DELAY_ANPASSUNG*(uint32_t)milliseconds);i++){
00088          }
00089 }
00090
00091 void __attribute__((__interrupt__, no_auto_psv)) _T1Interrupt(void)
00092     {
00093         _T1IF = 0; //Clear Timer1 interrupt flag
00094
00095         putsUART("Hello World\n");
00096
00097     }
00098
00099 //UART
00100 void initUART(){
00101     U1MODEbits.STSEL = 0; // 1-Stop bit
```

```
00102     U1MODEbits.PDSEL = 0; // No Parity, 8-Data bits
00103     U1MODEbits.ABAUD = 0; // Auto-Baud disabled
00104     U1MODEbits.UEN = 0;
00105     U1MODEbits.LPBACK = 0;
00106     U1MODEbits.RXINV = 0;
00107     //U1MODEbits.ALTIO = 0;
00108
00109     U1MODEbits.URXINV = 0;
00110     U1MODEbits.RTSMD = 0;
00111
00112     U1MODEbits.BRGH = 0; // Standard-Speed mode
00113     U1BRG = BRGVAL; // Baud Rate setting for 9600
00114
00115     U1STAbits.UTXISEL0 = 0; // Interrupt after one TX character is transmitted
00116     U1STAbits.UTXISEL1 = 0;
00117     U1STAbits.UTXBRK = 0;
00118     U1STAbits.ADDEN = 0;
00119     U1STAbits.UTXINV = 0;
00120     U1STAbits.URXISEL = 0;
00121     U1STA = U1STA | 0b0001000000000000;
00122     //_URXEN = 1;
00123
00124     //_U1RXIE = 1; // Enable UART RX interrupt
00125
00126     U1MODEbits.UARTEN = 1; // Enable UART
00127     //delay_ms(2);
00128     U1STAbits.UTXEN = 1; // Enable UART TX
00129
00130     /* Wait at least 105 microseconds (1/9600) before sending first char */
00131     delay_ms(2);
00132     _U1TXIE = 1; // Enable UART TX interrupt
00133
00134 }
00135
00136 void __attribute__((__interrupt__)) _U1TXInterrupt(void)
00137 {
00138     _U1TXIF = 0; // Clear TX Interrupt flag
00139
00140     getcFIFO_TX(&U1TXREG);
00141
00142 }
00143
00144
00145
00146
00147 int16_t putcFIFO_TX(char c)
00148 {
00149   //if (buffer.write >= BUFFER_SIZE)
00150   //  buffer.write = 0; // erhöht sicherheit
00151   _LATF0 = 1;
00152   if ( ( FIFO.write + 1 == FIFO.read ) ||
00153        ( FIFO.read == 0 && FIFO.write + 1 == BUFFER_SIZE ) )
00154     return BUFFER_FAIL; // voll
00155
00156   FIFO.data[FIFO.write] = c;
00157
00158   FIFO.write++;
00159   if (FIFO.write >= BUFFER_SIZE)
00160     FIFO.write = 0;
00161
00162   return BUFFER_SUCCESS;
00163 }
00164
00165 int16_t getcFIFO_TX(volatile uint16_t *c)
00166 {
00167   _LATF0 = 1;
00168   if (FIFO.read == FIFO.write)
00169     return BUFFER_FAIL;
00170
00171   *c = FIFO.data[FIFO.read];
00172
00173   FIFO.read++;
00174   if (FIFO.read >= BUFFER_SIZE)
00175     FIFO.read = 0;
00176
00177   return BUFFER_SUCCESS;
00178 }
00179
00180 int16_t putcUART(char c){
00181     _LATF0 = 1;
00182     _GIE = 0; // Interrupts ausschalten
00183     int16_t erfolg = putcFIFO_TX(c);
00184     _GIE = 1;
00185     return erfolg;
00186
00187
00188 }
```

```
00189
00190 int16_t putsUART(const char *str) {
00191     _LATF0 = 1;
00192     uint16_t i;
00193     uint16_t length = strlen(str);
00194
00195     _GIE = 0;   //Global Interrupt disable
00196     for(i = 0; i < length; i++) {
00197         //uint16_t ret = putcFIFO_TX(str[i]);
00198         if(! putcFIFO_TX(str[i]))
00199             break;
00200     }
00201     _GIE = 1;
00202     int16_t erfolg = -i;
00203     if(erfolg == -length)
00204         erfolg *= -1;
00205     _U1TXIF = 1; //Interuppt Routine Starten um FIFO-Inhalt zu senden
00206     return erfolg;
00207 }
00208
00209 //Timer1
00210 void init_timer1(){
00211     __builtin_write_OSCCONL(0b00000011); //SOSC aktivieren
00212     T1CONbits.TON = 0; // Disable Timer
00213     T1CONbits.TCS = 1; // Select external clock
00214     T1CONbits.TSYNC = 0; // Disable Synchronization
00215     T1CONbits.TCKPS = 0b00; // Select 1:1 Prescaler
00216     TMR1 = 0x00; // Clear timer register
00217     PR1 = 32767; // Load the period value, Quarztakt
00218
00219     IPC0bits.T1IP = 2; // Set Timer 1 Interrupt Priority Level
00220     IFS0bits.T1IF = 0; // Clear Timer 1 Interrupt Flag
00221     IEC0bits.T1IE = 1; // Enable Timer1 interrupt
00222     T1CONbits.TON = 1; // Start Timer
00223 }
00224
00225 //I2C
00226
00227 void Temp_FSM2(void)
00228 {
00229      static StateFunc statefunc = FSM2_Idle;
00230
00231      statefunc = (StateFunc)(*statefunc)();
00232 }
00233
00234 void initI2C(){
00235     I2C2CONbits.A10M = 0;
00236     I2C2BRG = 245; //100kHz
00237
00238     // Einschalten I2C mit eigenem Workaround, I2C Peripheriemodul kann hier leider nicht verwendet
      werden
00239     I2C_SDA_TRIS = 1;    // Pins wie einen Open-Kollektor-Treiber verwenden, d.h. 1 - hochohmig, 0
      wird getrieben
00240     I2C_SCL_TRIS = 1;
00241     I2C_SDA = 0;
00242     I2C_SCL = 0;
00243
00244         int j;
00245         for (j=0; j<=9; j++)    // takten bis min 1 Byte
00246         {
00247           I2C_SCL_TRIS = 0; delay_ms(1); // 5 us wÃ¤ren ausreichend ...100 kBaud
00248           I2C_SCL_TRIS = 1; delay_ms(1);
00249         }
00250         // Start Condition senden
00251         I2C_SCL_TRIS = 0;  delay_ms(1);
00252         I2C_SDA_TRIS = 0;  delay_ms(1);
00253         // Stop Condition senden
00254         I2C_SCL_TRIS = 1;  delay_ms(1);
00255         I2C_SDA_TRIS = 1;  delay_ms(1);
00256
00257         // Nun I2C erst anschalten
00258         _MI2C2IF = 0; //Interrupt falls noetig
00259         _MI2C2IE = 0;
00260         I2C2CONbits.I2CEN = 1;
00261
00262     //Sensor Pointer auf TEMP Register setzten
00263     I2C2CONbits.SEN=1; //start
00264     while(I2C2CONbits.SEN==1){}
00265
00266     //Tx Device address + Write bit
00267     I2C2TRN=0b10010000;
00268     while(I2C2STATbits.TRSTAT==1){}
00269
00270     if (I2C2STATbits.ACKSTAT==1){   //if NACK received, generate stop condition and exit
00271         I2C2STATbits.ACKSTAT=0;
00272         I2C2CONbits.PEN=1;
00273         while(I2C2CONbits.PEN==1){} //wait for the stop interrupt;
```

```
00274         return;
00275     }
00276
00277     //Tx Register Address
00278     I2C2TRN=0b00000000; //Pointer auf TEMP REGISTER setzten
00279     while(I2C2STATbits.TRSTAT==1){}
00280
00281     if (I2C2STATbits.ACKSTAT==1){   //if NACK received, generate stop condition and exit
00282         I2C2STATbits.ACKSTAT=0;
00283         I2C2CONbits.PEN=1;
00284         while(I2C2CONbits.PEN==1){} //wait for the stop interrupt;
00285         return;
00286     }
00287
00288     I2C2CONbits.PEN=1; //stop
00289     while(I2C2CONbits.PEN==1){} //wait for the stop interrupt
00290 }
00291
00292
00293
00294 void *FSM2_Idle(void)
00295 {
00296     static int c = 0;
00297     if (c>=999){
00298         c=0;
00299         return FSM2_Start;
00300     }
00301     c++;
00302     return FSM2_Idle;
00303
00304 }
00305
00306 void *FSM2_Start(void)
00307 {
00308     I2C2CONbits.SEN=1; //Start
00309     while(I2C2CONbits.SEN==1){}
00310     return FSM2_Adresse;
00311 }
00312
00313 void *FSM2_Adresse(void)
00314 {
00315     //Tx Device address + Read bit
00316     I2C2TRN=0b10010001;
00317     while(I2C2STATbits.TRSTAT==1){} //Warten solange übertragen wird
00318     return FSM2_ACK_Receive;
00319 }
00320
00321 void *FSM2_ACK_Receive(void)
00322 {
00323     if (I2C2STATbits.ACKSTAT==1){   //if NACK received, generate stop condition and exit
00324         I2C2STATbits.ACKSTAT=0;
00325         return FSM2_Stop;
00326     }
00327     return FSM2_Data_Receive;
00328 }
00329
00330 void *FSM2_Data_Receive(void)
00331 {
00332     int N=2; //2 bytes empfangen
00333     int i;
00334
00335     for(i=0;i<N;i++){
00336         I2C2CONbits.RCEN=1; //Empfangen aktivieren
00337         while(I2C2CONbits.RCEN==1){} //RCEN cleared automatically when SSP1IF goes high
00338
00339         data[i]=I2C2RCV;
00340
00341         //ACK sequence
00342         if (i<N-1){ I2C2CONbits.ACKDT=0; } //jedes byte mit ACK bestätigen
00343         else {I2C2CONbits.ACKDT=1;} //send NACK if this is the last Byte
00344
00345         I2C2CONbits.ACKEN=1; //start ack/nack sequence
00346         while(I2C2CONbits.ACKEN==1){}
00347
00348     } //end for loop
00349     return FSM2_Stop;
00350 }
00351
00352 void *FSM2_Stop(void)
00353 {
00354     I2C2CONbits.PEN=1;
00355     while(I2C2CONbits.PEN==1){} //wait for the stop interrupt
00356
00357     float temp = data[0]«8|data[1];
00358     char str[16];
00359     sprintf(str,"%f",temp/256);
00360     putsUART("Temperatur: ");
```

```
00361     putsUART(str);
00362     putsUART("°C");
00363     putsUART("\n");
00364 
00365     return FSM2_Idle;
00366 }
00367 
00368 
00369 
00370 /* Main Program                                                        */
00371 
00374 
00376 int16_t main(void)
00377 {
00378     DELAY_ANPASSUNG = ((SYS_FREQ/96)*2180ull)/1000000ull; //Berechnung der Delay Anpassung
00379     //uint16_t Count = 0;
00380     /* Configure the oscillator for the device */
00381     ConfigureOscillator();
00382     /* Initialize IO ports and peripherals */
00383     InitApp();
00384 
00385     //initUART();
00386     //init_timer1();
00387     init_ms_t4();
00388     //initI2C();
00389 
00390 
00391     TRISBbits.TRISB8 = 0; //LED0 als Ausgang
00392     ANSELBbits.ANSB8 = 0; //LED0 als Digitaler Ausgang
00393 
00394     TRISBbits.TRISB9 = 0; //LED als Ausgang
00395     ANSELBbits.ANSB9 = 0;
00396 
00397    //Taster als Eingänge
00398     _TRISG12 = 1;
00399     //Pull-up Widerstände einschalten
00400     _CNPUG12 = 1;
00401 
00402 
00403     _RP66R = _RPOUT_U1TX; //UART Pin Mapping
00404     RPINR18bits.U1RXR = 0b1011000;
00405     /* TODO <INSERT USER APPLICATION CODE HERE> */
00406 
00407     while(1)
00408     {
00409         PORTBbits.RB8=1;
00410         delay_ms(200);
00411         PORTBbits.RB8=0;
00412         delay_ms(200);
00413         //if(_T4IF)
00414         //{
00415             //_T4IF=0;
00416             //Count++;
00417             //if (Count >= HEARTBEAT_MS)
00418             //{
00419                 //Count = 0;
00420                 //Temp_FSM2();
00421             //}
00422 
00423         //}
00424     }
00425 }
```

5.11 system.c

```
00001 
00002 /* Files to Include                                                    */
00003 
00005 /* Device header file */
00006 #if defined(__XC16__)
00007     #include <xc.h>
00008 #elif defined(__C30__)
00009     #if defined(__dsPIC33E__)
00010         #include <p33Exxxx.h>
00011     #elif defined(__dsPIC33F__)
00012         #include <p33Fxxxx.h>
00013     #endif
00014 #endif
00015 
00016 
00017 #include <stdint.h>          /* For uint16_t definition                */
00018 #include <stdbool.h>         /* For true/false definition              */
00019 
```

```
00020 #include "system.h"          /* variables/params used by system.c             */
00021
00022
00023 /* System Level Functions                                                     */
00024 /*                                                                           */
00025 /* Custom oscillator configuration funtions, reset source evaluation         */
00026 /* functions, and other non-peripheral microcontroller initialization        */
00027 /* functions get placed in system.c.                                         */
00028 /*                                                                           */
00029
00031 /* Refer to the device Family Reference Manual Oscillator section for
00032 information about available oscillator configurations.  Typically
00033 this would involve configuring the oscillator tuning register or clock
00034 switching useing the compiler's __builtin_write_OSCCON functions.
00035 Refer to the C Compiler for PIC24 MCUs and dsPIC DSCs User Guide in the
00036 compiler installation directory /doc folder for documentation on the
00037 __builtin functions.*/
00038
00039 /* TODO Add clock switching code if appropriate.  An example stub is below.   */
00040 void ConfigureOscillator(void)
00041 {
00042     if (SYS_FREQ>7370000L){//Nur umschalten auf Primary wenn höhere Frequenz erwünscht
00043         switch (SYS_FREQ){
00044             case 8000000L:
00045                 //PLL muss nicht konfiguriert werden
00046                 // externer Quartz mit 8Mhz
00047                 break;
00048             case 50000000L:
00049                 CLKDIVbits.PLLPOST=2; //N2=4
00050                 PLLFBD=48; //M=50
00051                 CLKDIVbits.PLLPRE=0; //N1=2
00052                 break;
00053             case 70000000L:
00054                 CLKDIVbits.PLLPOST=2; //N2=4
00055                 PLLFBD=188; //M=190
00056                 CLKDIVbits.PLLPRE=3; //N1=5
00057                 break;
00058             case 100000000L:
00059                 CLKDIVbits.PLLPOST=0; //N2=2
00060                 PLLFBD=123; //M=125
00061                 CLKDIVbits.PLLPRE=3; //N1=5
00062                 break;
00063             case 140000000L:
00064                 CLKDIVbits.PLLPOST=0; //N2=2
00065                 PLLFBD=173; //M=175
00066                 CLKDIVbits.PLLPRE=3; //N1=5
00067                 break;
00068             //default:
00069                 //#error Tets
00070         }
00071         OSCTUN = 0;
00072
00073         if (SYS_FREQ == 8000000L){
00074             __builtin_write_OSCCONH(0x02); //Switch auf Primary ohne PLL
00075
00076             __builtin_write_OSCCONL(OSCCON | 0x01);
00077             while (OSCCONbits.COSC!= 0x02); //Warten bis gewechselt wurde
00078             }
00079
00080         else {
00081             __builtin_write_OSCCONH(0x03); //Switch auf Primary mit PLL
00082
00083             __builtin_write_OSCCONL(OSCCON | 0x01);
00084
00085             while (OSCCONbits.COSC!= 0x3); //Warten bis gewechselt wurde
00086             while (OSCCONbits.LOCK!= 1);
00087         }
00088
00089     }
00090 }
00091
00092
00093 //Timer1
00094 void init_timer1(){
00095     __builtin_write_OSCCONL(0b00000011); //SOSC aktivieren
00096     T1CONbits.TON = 0; // Disable Timer
00097     T1CONbits.TCS = 1; // Select external clock
00098     T1CONbits.TSYNC = 0; // Disable Synchronization
00099     T1CONbits.TCKPS = 0b00; // Select 1:1 Prescaler
00100     TMR1 = 0x00; // Clear timer register
00101     PR1 = 32767; // Load the period value, Quarztakt
00102
00103     IPC0bits.T1IP = 2; // Set Timer 1 Interrupt Priority Level
00104     IFS0bits.T1IF = 0; // Clear Timer 1 Interrupt Flag
00105     IEC0bits.T1IE = 1; // Enable Timer1 interrupt
00106     T1CONbits.TON = 1; // Start Timer
00107 }
```

```
00108
00109 void init_ms_t4(){
00110     T4CONbits.TON = 0; // Stop any 16/32-bit Timer3 operation
00111     T4CONbits.TCS = 0; // Select internal instruction cycle clock
00112
00113     T4CONbits.TGATE = 0; // Disable Gated Timer mode
00114     T4CONbits.TCKPS = 0b10; // Select 1:64 Prescaler
00115     TMR4 = 0x00; // Clear
00116     PR4 = (FCY/64000)-1; // Load 32-bit period value (lsw)
00117     //IFS0bits.T2IF = 0; // Clear Timer2 Interrupt Flag
00118     //IEC0bits.T2IE = 0; // Disable Timer2 interrupt
00119     T4CONbits.TON = 1; // Start 32-bit Timer
00120 }
```

5.12 system.h

```
00001
00002 /* System Level #define Macros                                          */
00003
00005 /* TODO Define system operating frequency */
00006
00007 /* Microcontroller MIPs (FCY) */
00008 //#define SYS_FREQ        7370000L
00009 //#define SYS_FREQ        8000000L
00010 #define SYS_FREQ        50000000L
00011 //#define SYS_FREQ        70000000L
00012 //#define SYS_FREQ        100000000L
00013 //#define SYS_FREQ        140000000L
00014
00015 #define FCY             SYS_FREQ/2
00016
00017
00018
00019 /* System Function Prototypes                                          */
00020
00022 /* Custom oscillator configuration funtions, reset source evaluation
00023 functions, and other non-peripheral microcontroller initialization functions
00024 go here. */
00025
00026
00027 //System Prototypen
00028 void ConfigureOscillator(void); /* Handles clock switching/osc initialization */
00029
00030 void init_timer1(void);
00031 void init_ms_t4(void);
00032
```

5.13 traps.c

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

```
00033 void __attribute__((interrupt,no_auto_psv)) _DMACError(void);
00034
00035 #endif
00036
00037 #if defined(__dsPIC33F__)
00038
00039 /* Use if INTCON2 ALTIVT=0 */
00040 void __attribute__((interrupt,no_auto_psv)) _AltOscillatorFail(void);
00041 void __attribute__((interrupt,no_auto_psv)) _AltAddressError(void);
00042 void __attribute__((interrupt,no_auto_psv)) _AltStackError(void);
00043 void __attribute__((interrupt,no_auto_psv)) _AltMathError(void);
00044
00045     #if defined(__HAS_DMA__)
00046
00047     void __attribute__((interrupt,no_auto_psv)) _AltDMACError(void);
00048
00049     #endif
00050
00051 #endif
00052
00053 /* Default interrupt handler */
00054 void __attribute__((interrupt,no_auto_psv)) _DefaultInterrupt(void);
00055
00056 #if defined(__dsPIC33E__)
00057
00058 /* These are additional traps in the 33E family.  Refer to the PIC33E
00059 migration guide.  There are no Alternate Vectors in the 33E family. */
00060 void __attribute__((interrupt,no_auto_psv)) _HardTrapError(void);
00061 void __attribute__((interrupt,no_auto_psv)) _SoftTrapError(void);
00062
00063 #endif
00064
00065
00066 /* Trap Handling                                                        */
00067 /*                                                                      */
00068 /* These trap routines simply ensure that the device continuously loops */
00069 /* within each routine.  Users who actually experience one of these traps */
00070 /* can add code to handle the error.  Some basic examples for trap code, */
00071 /* including assembly routines that process trap sources, are available at */
00072 /* www.microchip.com/codeexamples                                       */
00073
00075 /* Primary (non-alternate) address error trap function declarations */
00076 void __attribute__((interrupt,no_auto_psv)) _OscillatorFail(void)
00077 {
00078         INTCON1bits.OSCFAIL = 0;        /* Clear the trap flag */
00079         while(1);
00080 }
00081
00082 void __attribute__((interrupt,no_auto_psv)) _AddressError(void)
00083 {
00084         INTCON1bits.ADDRERR = 0;        /* Clear the trap flag */
00085         while (1);
00086 }
00087 void __attribute__((interrupt,no_auto_psv)) _StackError(void)
00088 {
00089         INTCON1bits.STKERR = 0;         /* Clear the trap flag */
00090         while (1);
00091 }
00092
00093 void __attribute__((interrupt,no_auto_psv)) _MathError(void)
00094 {
00095         INTCON1bits.MATHERR = 0;        /* Clear the trap flag */
00096         while (1);
00097 }
00098
00099 #if defined(__HAS_DMA__)
00100
00101 void __attribute__((interrupt,no_auto_psv)) _DMACError(void)
00102 {
00103         INTCON1bits.DMACERR = 0;        /* Clear the trap flag */
00104         while (1);
00105 }
00106
00107 #endif
00108
00109 #if defined(__dsPIC33F__)
00110
00111 /* Alternate address error trap function declarations */
00112 void __attribute__((interrupt,no_auto_psv)) _AltOscillatorFail(void)
00113 {
00114         INTCON1bits.OSCFAIL = 0;        /* Clear the trap flag */
00115         while (1);
00116 }
00117
00118 void __attribute__((interrupt,no_auto_psv)) _AltAddressError(void)
00119 {
00120         INTCON1bits.ADDRERR = 0;        /* Clear the trap flag */
```

```
00121         while (1);
00122 }
00123
00124 void __attribute__((interrupt,no_auto_psv)) _AltStackError(void)
00125 {
00126         INTCON1bits.STKERR = 0;         /* Clear the trap flag */
00127         while (1);
00128 }
00129
00130 void __attribute__((interrupt,no_auto_psv)) _AltMathError(void)
00131 {
00132         INTCON1bits.MATHERR = 0;        /* Clear the trap flag */
00133         while (1);
00134 }
00135
00136     #if defined(__HAS_DMA__)
00137
00138     void __attribute__((interrupt,no_auto_psv)) _AltDMACError(void)
00139     {
00140          INTCON1bits.DMACERR = 0;        /* Clear the trap flag */
00141          while (1);
00142     }
00143
00144     #endif
00145
00146 #endif
00147
00148
00149 /* Default Interrupt Handler                                              */
00150 /*                                                                        */
00151 /* This executes when an interrupt occurs for an interrupt source with an     */
00152 /* improperly defined or undefined interrupt handling routine.                */
00153
00154 void __attribute__((interrupt,no_auto_psv)) _DefaultInterrupt(void)
00155 {
00156         while(1);
00157 }
00158
00159 #if defined(__dsPIC33E__)
00160
00161 /* These traps are new to the dsPIC33E family.  Refer to the device Interrupt
00162 chapter of the FRM to understand trap priority. */
00163 void __attribute__((interrupt,no_auto_psv)) _HardTrapError(void)
00164 {
00165     while(1);
00166 }
00167 void __attribute__((interrupt,no_auto_psv)) _SoftTrapError(void)
00168 {
00169     while(1);
00170 }
00171
00172 #endif
```

5.14 user.c

```
00001
00002 /* Files to Include                                                       */
00003
00005 /* Device header file */
00006 #if defined(__XC16__)
00007     #include <xc.h>
00008 #elif defined(__C30__)
00009     #if defined(__dsPIC33E__)
00010         #include <p33Exxxx.h>
00011     #elif defined(__dsPIC33F__)
00012         #include <p33Fxxxx.h>
00013     #endif
00014 #endif
00015
00016 #include <stdint.h>          /* For uint16_t definition                   */
00017 #include <stdbool.h>         /* For true/false definition                 */
00018 #include "user.h"            /* variables/params used by user.c           */
00019
00020
00021 /* User Functions                                                         */
00022
00024 /* <Initialize variables in user.h and insert code for user algorithms.> */
00025
00026 void InitApp(void)
00027 {
00028     /* TODO Initialize User Ports/Peripherals/Project here */
00029
00030     /* Setup analog functionality and port direction */
```

```
00031
00032     /* Initialize peripherals */
00033 }
00034
00035 void setLED(uint16_t nr)
00036 {
00037    if (nr>=4) return;
00038    LATB = LATB | (1 « nr+8);
00039 }
00040
00041
00042
00043 //4-bit Wort -> RB8-11
00044
00045 //uint16_t leds=0b 0000 0000 0000 1101;
00046
00047 //LATB = (LATB & ~0b0000111100000000) | ((leds«8) &0b0000111100000000);
```

5.15 user.h

```
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 /* TODO Application specific user parameters used in user.c may go here */
00013
00014
00015 /* User Function Prototypes                                                   */
00016
00018 /* TODO User level functions prototypes (i.e. InitApp) go here */
00019
00020 void InitApp(void);         /* I/O and Peripheral Initialization */
00021
00022 void setLED(uint16_t nr);
```

Index