Lucerne University of
Applied Sciences and Arts

HOCHSCHULE
LUZERN

Microcontroller

Testatprojekt 2

Horw, 15.12.2009

Projekt	Testatprojekt 2
Dokument	Projektdokumentation
Schule	Hochschule Luzern, Technik & Architektur
Modul	TA.MC.H0901
Projektteam	**Galliker Thomas** Studiengang Informatik (BB) Panorama 6123 Geiss Tel. +41 79 504 80 70 thomas.galliker@stud.hslu.ch **Moser Christoph** Studiengang Informatik (BB) Zugerbergstrasse 41 6314 Unterägeri Tel. +41 79 785 19 07 christoph.moser@stud.hslu.ch
Dozenten	**Prof. Jost Christian**
Letzte Änderung	15. Dezember 2009, 21:34:00 Uhr

Änderungsprotokoll

Version	Datum	Autor	Beschreibung
0.1	05.12.2009	gat	Dokument erstellt
0.2	11.12.2009	moc	Vorgehen / Fehler dokumentiert
0.3	12.12.2009	moc	Lessons Learned angefügt
0.4	14.12.2009	gat	Überarbeitung und Logic Analysis
0.5	15.12.2009	gat	Korrekturen und Abgabe

Inhalt

Abbildungsverzeichnis

1 Ausgangslage

Aufgrund des erhaltenen Blockdiagramms und des bestehenden Codes, soll ein LCD-Wecker basierend auf dem I²C-Bussystem entworfen werden. Die Hardware und Software, der Aufgabenstellung kann Fehler enthalten. Es sollen mindestens fünf Fehler ermittelt und korrigiert werden.

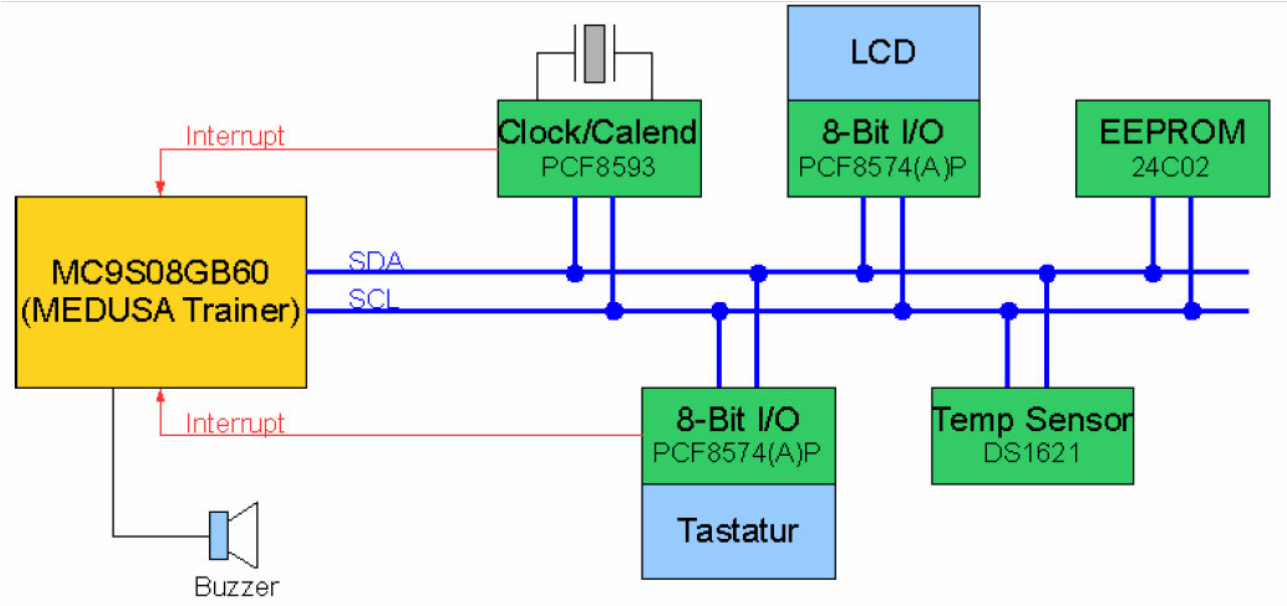

Abbildung 1: Systemarchitektur der LCD-Wecker Applikation (Quelle: Aufgabenstellung)

2 Vorgehen und Arbeitsaufteilung

Es wurde systematisch nach den Fehlern gesucht. Im ersten Schritt wurde die Hardware anhanden des vorgegebenen Schemas aufgebaut. Danach wurde anhanden der Datensheets die Verkablung an den IC's überprüft. Dabei haben wir festgestellt, dass die SCL und SDA Leitung am Alarm Clock und am EEPROM vertauscht waren. Da nach dem vertauschen der SDA und SCL Leitungen, nach wie vor keine Anzeige auf dem Display erschien, entschieden wir uns mit der Theorie des I²C-Bus (anhanden des MC-Skript) vertraut zu machen. Beim durchlesen der Theorie haben wir festgestellt, das die Pull-Up Widerstände am I²C-Bus fehlten. Auch nachdem wir die Pull-Up Widerstände korrekt angeschlossen haben, war auf dem Display nichts ersichtlich. Nun wurde die Kommunikation zwischen den einzelnen Hardwarekomponenten überprüft, dabei wurden die ersten Fehler im Code gefunden. Weitere Fehler wurden mit Hilfe des Debuggers und der Funktionsbeschreibung der Aufgabenstellung gefunden. Die gefunden Fehler waren nun funktionsbezogen und konnte im Team aufgeteilt werden, so dass jeder sich mit der Codeumsetzung einer Weckerfunktion befassen konnte. Am Ende wurden die überarbeiteten Code-Teile in einem Programm zusammengefasst. Parallel wurden die Erkenntnisse die bei der Analyse gefunden wurden, in diesem Dokument zusammengetragen.

3 Fehleranalyse

In diesem Abschnitt die gefunden Fehler und die dazugehörigen Lösungen dokumentiert.

3.1 I²C-Busanschluss

Die Taktleitung SCL und die Datenleitung SDA des I²C Busses sind auf dem Schema verkehrt an den IC's Digital Thermometer (DS1621) und EEPROM (M24C02) angeschlossen. Die Korrektur wurde in roter Farbe in die nachfolgende Abbildung eingebracht:

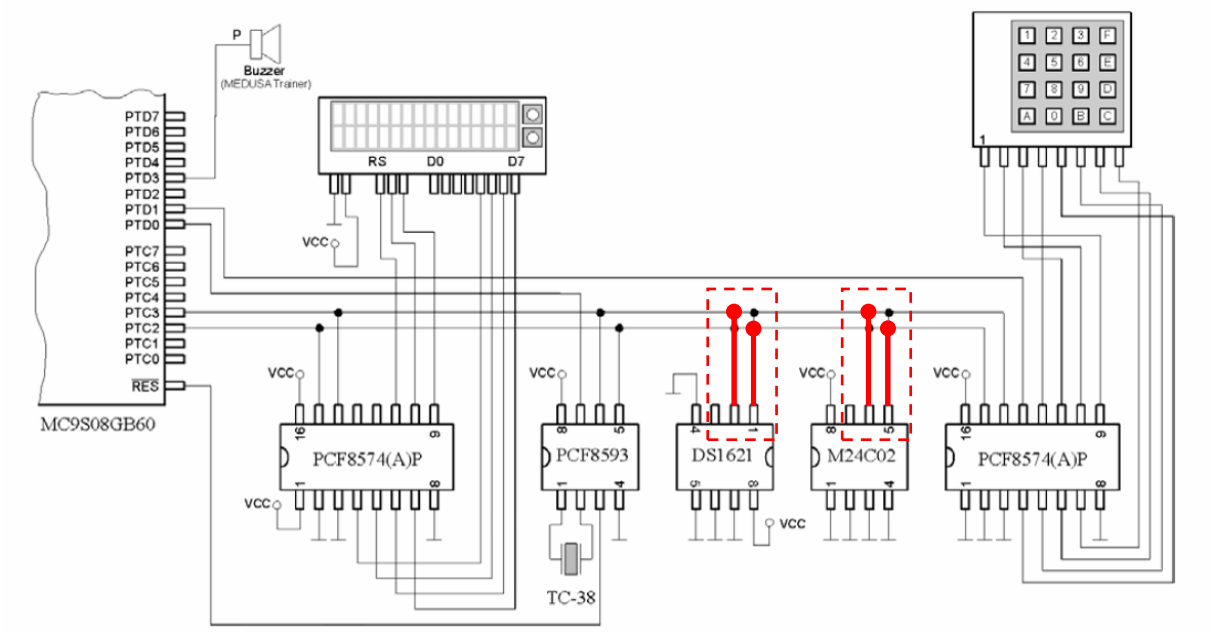

Abbildung 2: Elektroschema der LCD-Wecker Applikation (Quelle: Aufgabenstellung)

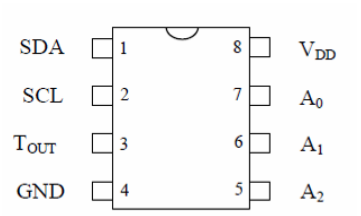

Abbildung 3: Digital Thermometer DS1621

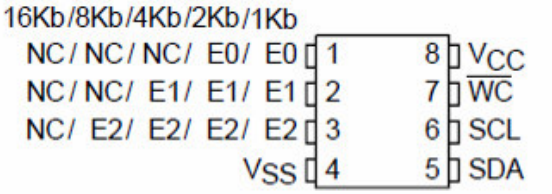

Abbildung 4: EEPROM M24C02

3.2 Pull-up für I²C-Bus

SDA und SCL sind bidirektionale Leitungen die über einem Pull-up Widerstand mit der positiven Versorgungsspannung verbunden werden müssen. Im Schema sind die Pull-up Widerstände nicht enthalten.

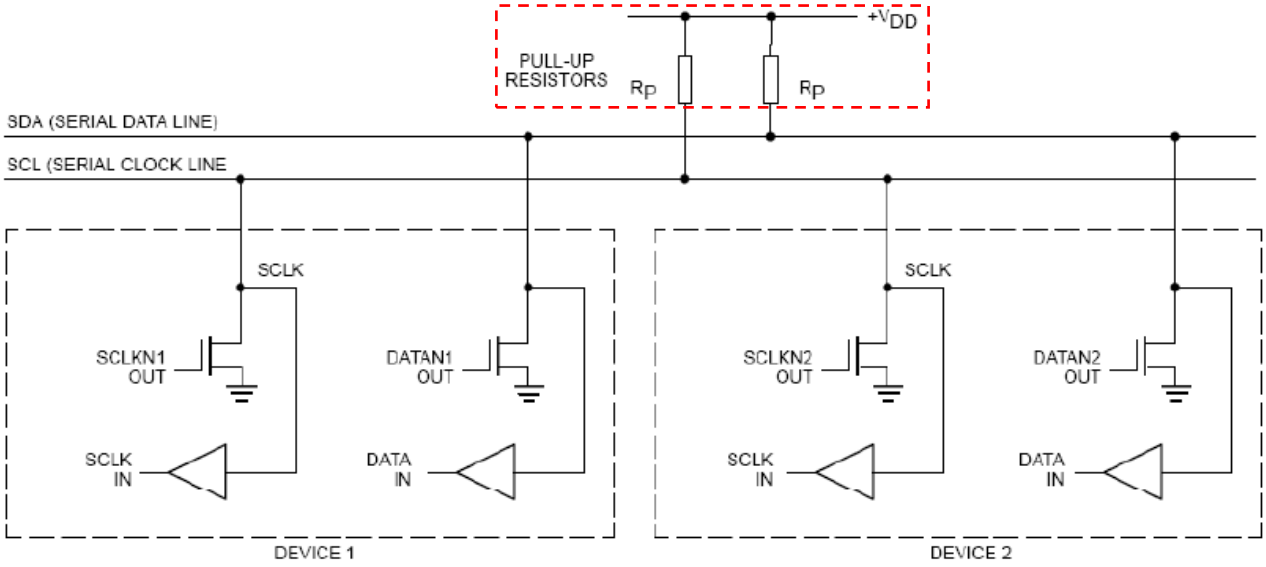

Abbildung 5: Prinzipdarstellung des IIC Bus (Quelle: MC-Script)

3.3 I²C Bus Adressen

Die Überprüfung der definierten Busadressen der I²C Busteilnehmer im Source Code hat ergeben, dass die Adressen der zwei Remote 8-bit I/O Expander (PCF8574) nicht mit den Adressen im Datesheet übereinstimmen. Es muss beachtet werden, dass der feste Adressteil der Chiptypen PCF8574P und PCF8574AP nicht identisch sind und im Datenblatt nicht sehr klug beschriftet sind.

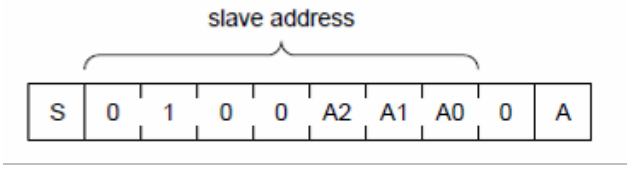

Abbildung 6: Adressierung vom Chiptypen PCF8574P

3.3.1 Berechnen der I²C Adressierung für das Keyboard

Die Adresse für den IC Chip, welcher das Keyboard ansteuert, ist in IC2_ADR_KEYBOARD definiert und setzt sich wie folgt zusammen:

- Fixe Adressierung für den Chip PCF8574P: 0100…0
- Frei wählbare Adresse:
  A2: 0
  A1: 0
  A0: 0

Daraus folgt: 01000000 → 0x40. Folgende Änderungen mussten im Code vorgenommen werden:

```
// I2C-Adressen der I2C-Bausteinen
#define I2C_ADR_TEMP          0x90
#define I2C_ADR_EEPROM        0xA0
#define I2C_ADR_KEYBOARD      0x40   //instead of 0x46
#define I2C_ADR_RTC           0xA2
#define I2C_ADR_LCD           0x42
```

3.3.2 Berechnen der I²C Adressierung für das LCD Display

Die Adresse für den IC Chip, welcher das LCD Display ansteuert, ist in I2C_ADR_LCD definiert und setzt sich wie folgt zusammen:

- Fixe Adressierung für den Chip PCF8574P: 0100…0
- Frei wählbare Adresse:
  A2: 0
  A1: 0
  A0: 1

Daraus folgt: 01000010 → 0x42. Folgende Änderungen mussten im Code vorgenommen werden:

```
// I2C-Adressen der I2C-Bausteinen
#define I2C_ADR_TEMP          0x90
#define I2C_ADR_EEPROM        0xA0
#define I2C_ADR_KEYBOARD      0x40
#define I2C_ADR_RTC           0xA2
#define I2C_ADR_LCD           0x42   //instead of 0x44
```

3.4 Falscher Interruptvektor in Project.prm

Beim Debuggen wurde festgestellt, dass der Interrupt für den Timer1/Channel1 nicht gesetzt war. Das Programm blieb immer in der folgenden endlosen for-Schlaufe hängen.

```
interrupt void errISR_TPM1CH1() { for (;;)  asm BGND; }
```

Dies wurde korrigiert indem der Interruptvektor im File „Project.prm“ mit dem richtigen Label ersetzt wurde.

```
VECTOR ADDRESS 0xFFF2 isrTPM1CH1    // TPM1 channel 1
```

3.5 Alarm Reset

Nachdem ein Alarm ausgelöst wurde, sollte dieser mit einer beliebigen Taste des Keyboards bestätigt werden. Beim Bestätigen wird die Funktion `rtcResetAlarm` ausgeführt. In diesem Code wird das `AlarmFlag` des Kontroll-Registers zurückgesetzt, damit nach einem aufgetretenen Alarm, der Alarm Interrupt erneute aufgerufen werden kann.

```
/**
 * Setzt den Alarm zurück, indem das Alarm-Flag zurückgesetzt wird
 *
 * @return
 *    EC_SUCCESS im Erfolgsfall, ansonsten ein Fehlercode
 */
unsigned char rtcResetAlarm(void)
{
    unsigned char ucResult;
    tControlStatusReg ctrlReg;

    ctrlReg.Byte = 0;      // Uhr konfigurieren (Alarm-Funktionalität aktivieren)
    ctrlReg.Bits.alarmFlag = 0;   // Alarm-Flag zurücksetzen
    ctrlReg.Bits.alarmEnableBit = 1;    // Alarm Konrollregister aktivieren

    ucResult = i2cWriteCmdData(i2cAdr, REG_CONTROL_STATUS, &ctrlReg.Byte, 1);
    if (ucResult != EC_SUCCESS) return ucResult;

    return EC_SUCCESS;
}
```

3.6 Temperaturumschaltung Celcius / Fahrenheit

Beim Betätigen der Taste „0“ wurde die Temperatur nicht auf Fahrenheit umgeschaltet. Beim Analysieren des Source Codes wurde festgestellt, dass die Umrechnung für Fahrenheit nicht realisiert war. Für die Umrechnung von Celsius nach Fahrenheit wurde folgende Formel verwendet:

```
Fahrenheit = Celcius * 5 / 9 + 320
```

Da die Temperatur durch den DS1621 in Grad Celsius ausgelesen wird, ist das Umrechnen von Fahrenheit nach Celsius nicht nötig. Die letzte Stelle des Temperaturwerts wird als Nachkommastelle interpretiert.

```
void tempGetCurrentTempString(char *str, bool celsius)
{
    int16 tmp;
    bool tempIsPositive;
    (void)celsius;

    str[3] = '.';
    str[5] = 0xDF; // str[5] = '°';

    if (temp == TEMP_INVALID)
    {
        str[0] = '-';
        str[1] = '-';
        str[2] = '-';
        str[4] = '-';
        return;
    }
    if(celsius)              // Prüfen ob Celcius oder Fahrenheit angezeigt werden soll
    {
       str[6] = 'C';                      // C für Celsius ausgeben
       tmp = temp;                        // Keine Umrechnung bei Celsius Ausgabe nötig
    }
    else
    {
       str[6] = 'F';                       // F für Fahrenheit ausgeben
       tmp = temp * 9 / 5 + 320;           // Umrechnung von Celcius nach Fahrenheit
    }
```

Nachfolgende Abbildungen zeigen die Funktionsweise. Beim Betätigen der Taste „0“ kann nun zwischen °C und °F umgeschaltet werden. Die Umrechnungsresultate wurden kontrolliert und sind korrekt.

Abbildung 7: Temperaturanzeige in °C

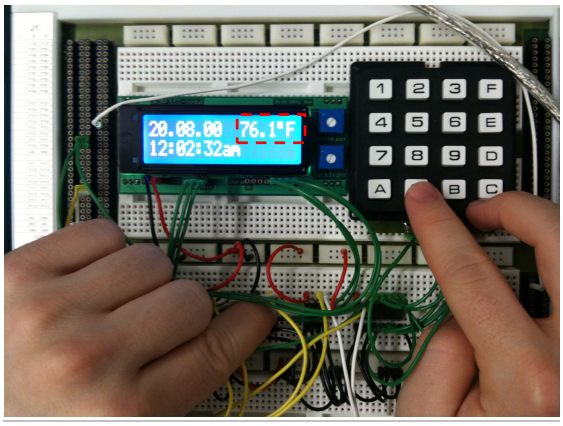

Abbildung 8: Temperaturanzeige in °F

3.7 Weckzeit / Datum speichern

Die Weckzeit und das Datum wird beim erhaltenen Code nicht gespeichert, respektive bei einem Neustart des Weckers nicht korrekt ausgelesen. Dieses Problem ist aufgetreten, da der Wecker beim Speicher und Laden nicht auf die gleiche Adresse im EPROM zugreift.

```
unsigned char memWriteBuf(unsigned char *ucBuf, unsigned char ucLength)
{
    return i2cWriteCmdData(i2cAdr, BYTE_ADR+1, ucBuf, ucLength);
}
```

Die Uhrzeit kann nicht gespeichert werden, da der Wecker über keinen Standby-Zustand implementiert. Um dies zu realisieren, müsste der Wecker die Uhr auch im Ruhezustand weiter ticken lassen.

3.8 Buzzer Fehler

Während der Alarm auf dem Wecker ausgeführt wurde, war kein Alarm Buzzer hörbar. Das Problem beruhte in der Funktion alarmClockTimerCallback. Diese Funktion ist zuständig für den Aufruf der Funktion buzzerBeep, welche den Alarmton generiert. Die Parameter im Funktionsaufruf buzzerBeep waren vertauscht:

```
buzzerBeep(BUZZER_PULSE, BUZZER_FREQUENCY);
```

…wurde ersetzt durch…

```
buzzerBeep(BUZZER_FREQUENCY, BUZZER_PULSE);
```

Dies ergibt den folgenden Code für die Methode alarmClockTimerCallback():

```
void alarmClockTimerCallback(tTimerID timerID)
{
    static unsigned char alarmCounter = DISPLAY_REFRESH_RATE;
    (void)timerID;

    if (alarmEvent)
    {                                       // Falls Wecker ausgelöst hat...
        alarmCounter--;                     // alarmCounter dekrementieren
        if (alarmCounter == 0)
        {
            // 1x pro Sekunde, Buzzer aktivieren
            buzzerBeep(BUZZER_FREQUENCY, BUZZER_PULSE);
            dispAlarm = !dispAlarm;         // "ALARM" auf dem LCD blinken lassen
            alarmCounter = DISPLAY_REFRESH_RATE;        // Delay für 1sec
        }
    }
    else
    {
        dispAlarm = FALSE;                              // Weckzeit noch nicht erreicht
        alarmCounter = DISPLAY_REFRESH_RATE;            // Variablen zurücksetzen
    }
    // Falls der Benutzer sich nicht in einem Menü befindet
    if (alarmClockState == idle) alarmClockUpdate();
}
```

4 Logikanalyse

Nach erfolgter Korrektur der I²C-Fehler konnte die erfolgreiche Kommunikation mit Hilfe des Logic Analyzers festgestellt werden. Dazu wurden beide Busleitungen, SCL und SDA, sowie das Nullpotenzial GND mit dem Logic Analyzer verbunden. Die Taktleitung SCL wurde am Logic Analyzer an Channel 3 angeschlossen, während für die Datenleitung SDA der Channel 4 herhalten musste. Ein Trigger auf Channel 4 (SDA) reagiert auf sinkende Flanken (1 zu 0). Auf der nachfolgenden Abbildung sind Start- und Stop-Signale der Datenübertragung gut ersichtlich. Das Start-Signal ist eine fallende Flanke auf SDA während SCL high ist, das Stop-Signal ist eine steigende Flanke auf SDA während SCL high ist.

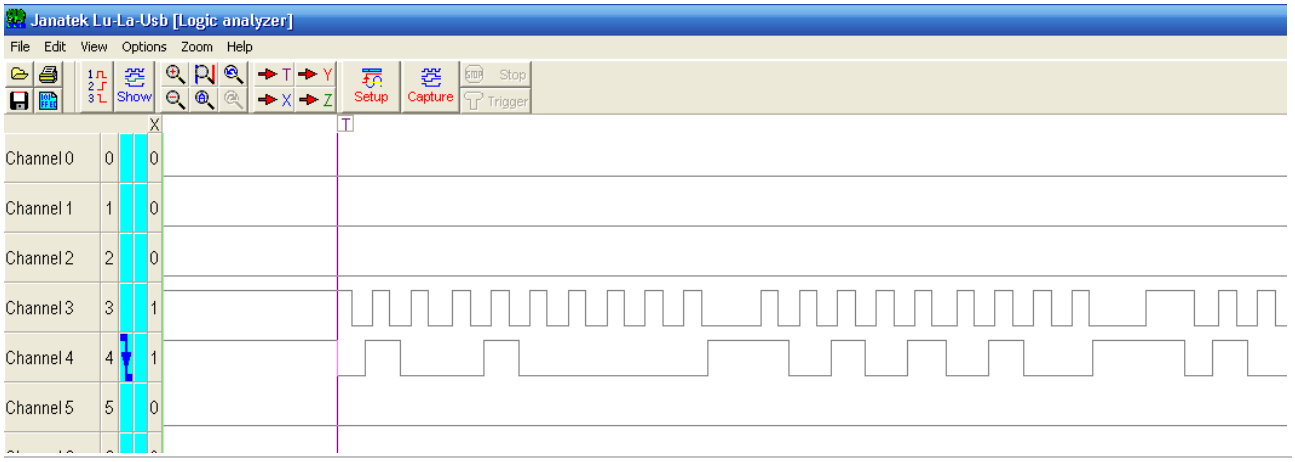

Abbildung 9: Logic Analysis eines I²C Datentransfers

Welche Bedeutung die übertragenen Daten haben ist nicht feststellbar, da zum Zeitpunkt der Aufzeichnung viele verschiedene Informationen über der I²C Bus übertragen wurden. Wir gehen davon aus, dass es sich um Display-Refresh bzw. Zeit-Update Informationen handelt.

5 Lessons Learned

- Es lohnt sich zuerst die Theorie des I²C-Bus zu verstehen, bevor man den Code analysiert, da der I²C-Bus einige Stolpersteine (Pull-Up, Start-Bit, Stop-Bit, Adressierung der IC's) hat, die man beachten muss.
- Die Tonausgabe von PTD3 muss an den Input-Port des Lautsprechers ausgegeben werden. Fragt sich nur, für was ein Lautsprecher ein Output-Port besitzt?
- Der Kontrast des LCD-Displays muss korrekt eingestellt werden. Ansonsten kann man lange nach Displayfehler suchen, wenn das Angezeigte nicht sichtbar ist.
- Die Zeit AM / PM wechselt bei 12:59 auf 01:00 zurück, sprich dies ist kein Fehler und muss somit nicht korrigiert werden.
- Im 12h-Zeitmodus wechselt die AM/PM Anzeige um 12:59 a.m. nach 01:00 p.m. (Dasselbe gilt für den Wechsel von p.m. zu a.m.). Das war sehr verwirrend und völlig entgegen unserer Meinung. Schliesslich mussten wir eingestehen, dass wir einen „falschen Fehler" gefunden haben.
- Eine systematische Vorgehensweise ist das A und O bei der erfolgreichen Fehlersuche in komplexen Systemen.