Erfahrungen aus der Erstellung des Projektes „ThermoAmpel“ für das Seminar „Mikrocontrollerschaltungen - Realisierung in Hard- und Software“

Ilhan Aydin, Lev Perschin

26. November 2018

Zusammenfassung

1 Erste Erfahrungen/Schritte

Wir, Ilhan und Lev, hatten wenig Vorwissen in diesem Bereich. Wir haben noch nie ein Schaltplan erstellt, die Komponenten auf dem Board verteilt, es gedruckt, bestückt und anschließend programmiert. Das Wissen, welches wir hatten, waren Kenntnisse aus der Vorlesung „Technische Informatik“ und ähnliche.

2 Schaltplan

- Auswahl der richtigen Teile - größtenteils von PDF übernommen
- Berechnung von LED Widerstanden
- Übersichtlichkeit wichtig
- Offene Pins mit einem Stecker(?) versehen, damit man sie in Zukunft leichter verwenden kann

3 Board

- Positionierung der Teile - im Nachhinein vielleicht wie es am schlausten wäre?
- VDD Leitung etwas dicker
- Groundpins mit Verbindung zur Groundplate richtig einstellen
- Nach dem Löten mit Messgerät schauen, ob Verbindungen da sind wo sie sein sollten und nicht da sind wo sie nicht sein sollten

4 Funktionsweise der Software

Dieser Abschnitt beschreibt unsere Herangehensweise an das Programmieren des Mikrocontrollers und die Funktionsweise des Programms. Ausgehend davon, dass in der Platine alles korrekt verschaltet ist, kann man mit dem Programmieren beginnen. Wir haben dafür einen Programmer von DIAMEX verwendet. Vorteil bei diesem Programmer ist, dass es die Platine mit 5V versorgen kann, sodass man beim Programmieren die Platine nicht extra an eine Stromversorgung anschließen muss. ***-HIER ROUTINE ZUM ÜBERPRÜFEN DER ATMELSTUDIO ERKENNUNG -> Probleme mit 5V usw -> Erdungspins nicht korrekt -> wurde ersichtlich, nach dem schreiben des LED programms.*** Zum Testen, ob der Mikrocontroller mit unserem Programm überschrieben worden ist, haben wir vorerst ein einfaches Programm geschrieben, welches die LEDs ansteuert. Nachdem die oben genannten Schwierigkeiten überwunden waren und der Mikrocontroller das LED-Programm richtig ausgeführt hat, begannen wir mit der Implementation unserer Aufgabe. **vielleicht etwas weiter am anfang

aufgabe beschreiben?**Die Aufgabe bestand daraus mittels eines Temperatur- und Luftfeuchtigkeitsensors die Temperatur und Luftfeuchtigkeit auszulesen und über die UART-Schnittstelle auszugeben. Zusätzlich lassen wir über die LEDs anzeigen, ob die Temperator und Luftfeuchtigkeit im guten Bereich liegen.

4.1 Aufbau des Programms

In diesem Abschnitt wird der Aufbau des Programms beschrieben.
Vor allem Anderen sollte man "m16adef.inc"über `.include` einbinden, damit man für den Mikrocontroller spezifische Definitionen verwenden kann. Danach sollte man um das Programm lesbarer zu machen, häufig verwendete Register oder Register die nur einen bestimmten Zweck haben über `.def` Namen geben. Ebenfalls nützlich ist auch die Verwendung von `.equ`, womit man häufig verwendeten Werten Namen geben kann.
Durch `.cseg` gibt man an, dass man sich ab der Zeile im Codesegment befindet. Als Erstes sollte hier die Startadresse stehen. Zu dieser sollte man beim Start oder Reset des Mikrocontrollers springen. Dies erreicht man mit `.org` für die Adresse 0x000. Dort gibt man dann an, wo das Programm beginnen soll.
Bevor es in die Hauptroutine geht, legen wir fest, was beim Starten oder Neustarten des Mikrocontrollers passieren soll. Zunächst wird der Stack initialisiert. Danach legen wir die LED Pins als Ausgang fest. Weiterhin aktivieren und konfigurieren hier die Interrupts. Zudem wollen wir den 8-Bit und 16-Bit Timer verwenden, welche hier ebenfalls konfiguriert werden. Zum Schluss initialisieren wir noch die UART Schnittstelle.
Die Hauptroutine ist eine Endlosschleife und besteht aus der ständigen Abfrage der Temperatur und relativen Luftfeuchtigkeit gekoppelt mit einem kleinen Wetterbericht, welcher die abgefragten Werte über die UART Schnittstelle überträgt. Zwischen jeder Abfrage haben wir einen Delay von einer Sekunde eingebaut, damit der Sensor nicht überbelastet wird.

4.2 Timer/Counter

4.3 Interrupts

Für die Aktivierung von globalen Interrupts muss man zunächst das I Bit im `SREG` setzen. Der Befehl `sei` macht genau dies. Um die Interrupts bei den Pins INT0, INT1 und INT2 nun zu aktivieren, setzen wir im `GICR` Register die Bits `INT0`, `INT1` und `INT2` auf 1. Damit ein Interrupt beim Drücken des Knopfes erzeugt wird, muss eingestellt werden, dass ein Interrupt für INT0-INT2 bei einer fallenden Flanke erzeugt wird. Dies stellt man im `MCUCR` Register ein, indem man für INT0 das Bit `ISC01` und für INT1 das Bit `ISC11` auf 1 setzt. Für INT2 werden standardmäßig Interrupts bei einer fallenden Flanke erzeugt. Um jetzt auf Interrupts zu warten, setzt man die Pins INT0, INT1, INT2 auf Eingang und legt eine 1 an.
Handler**

4.4 Ansteuern des Sensors

4.4.1 Wie man es nicht machen sollte:

Zunächst haben wir probiert, mit dem Sensor über das Two-wire Serial Interface (TWI) zu kommunizieren. Wie man über TWI kommuniziert ist im Datenblatt des Atmega16 ausführlich mit Codebeispielen beschrieben ***(Seite 175 ff.)***. Nach der Implementierung einer Routine zum Kommunizieren über TWI fiel uns auf, dass der Sensor TWI gar nicht unterstützt. Eine andere Lösung musste her.

4.4.2 Wie man es machen sollte:

Wie man mit dem Sensor richtig kommuniziert, steht im Datenblatt des Sensors geschrieben. Dazu sendet man dem Sensor ein Kommando über den SCK- und DATA-Pin des Sensors, wartet auf die Bearbeitung des Kommandos und liest zum Schluss die Antwort aus. Das Senden eines Kommandos wird mit über das Senden der folgenden Bitfolge initiiert, welche im Datenblatt auch „Transmission Start“ genannt wird:

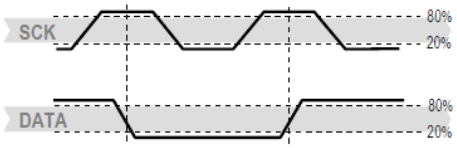

Abbildung 1: Transmission Start

Um das Senden im Programm zu realisieren setzt man zunächst die Pins, über die SCK und DATA verbunden sind, auf Ausgang. Dann setzt man bei DATA eine 1 und bei SCK eine 0 an, damit der Zustand von DATA mit High und der von SCK mit Low beginnt, wie auch in der Abbildung dargestellt ist. Wie man weiter vorgeht, kann man der Abbildung entnehmen. Wenn der Zustand vom SCK oder DATA von Low auf High wechselt, bedeutet das, dass man an den mit SCK oder DATA verbundenen Port eine 1 anlegt. Der Wechsel von High zu Low entspricht dem anlegen einer 0. Nachdem man Transmission Start übertragen hat, kann man nun ein Kommando übertragen. Um beispielsweise die Temperatur abzufragen, sendet man die Sequenz 00000011. Das Senden geschieht ebenfalls durch das Setzen und Entfernen von Bits am Port von SCK und DATA. Wenn der Sensor ein Kommando erhalten hat, sendet er ein ACK zurück. Zum Überprüfen, ob das Senden des Kommandos erfolgreich war, setzt man DATA auf Eingang und legt eine 1 an, um den internen Pullup-Widerstand zu nutzen. Dies ist nötig, um die vom Sensor gesendeten Bits lesen zu können. Wird DATA vom Sensor aus auf Low gesetzt, hat er das Kommando erhalten. Nun wartet man bis die Messung fertig ist. Der Sensor setzt DATA auf Low, wenn es die Messung beendet hat. Das Lesen der Daten erfolgt Byteweise. Nachdem man ein Byte gelesen hat, sendet man dem Sensor ein ACK und liest

danach das nächste Byte aus, falls die Daten eine höhere Genauigkeit als acht Bit haben. Insgesamt liest man drei Bytes aus, wobei letzte Byte die Checksumme beinhaltet. Die Kommunikation endet nach dem Auslesen des dritten Bytes. Man kann sie auch nach dem Lesen des zweiten Bytes beenden, wenn man die Checksumme nicht benötigt. Dazu lässt man DATA nach dem Senden des ACKs auf High.
Beim Kommunizieren ist es wichtig, nicht zu vergessen, dass wenn man etwas senden will, der Port von DATA auf Ausgang und beim Lesen auf Eingang gesetzt werden muss. Zudem sollte man die Häufigkeit der Anfragen auf eine pro Sekunde beschränken, um die vom Sensor abgegebene Wärme zu reduzieren.

4.5 Auswerten der Daten

Die Daten, die man vom Sensor erhält, müssen für die Ausgabe umgewandelt werden. Dazu gibt es im Datenblatt des Sensors Formeln, wie zum Beispiel Folgendes für die Berechnung der relativen Luftfeuchtigkeit:

$$RH_{linear} = c_1 + c_2 * SO_{RH} + c_3 * SO_{RH}^2 \,(\%RH)\,,$$

wobei SO_{RH} die vom Sensor ausgelesenen Bits bezeichnet. Wenn SO_{RH} eine Genauigkeit von 12 Bit hat, verwendet man für die Konstanten c_1, c_2 und c_3 folgende Werte: $c_1 = -2.0468$, $c_2 = 0.0367$, $c_3 = -1.5955 * 10^{-6}$. Beispielsweise würde die vom Sensor erhaltene Bitfolge "0000 0100 0011 0001"(1073) einer relativen Luftfeuchtigkeit von 35.50% entsprechen.
Die Formel kann man allerdings nicht im Mikrocontroller ohne Weiteres verwenden, da diese Genauigkeiten zum Berechnen erfordert, die der Mikrocontroller nicht aufbringt. Eine Möglichkeit zum Umwandeln der Bits in relative Luftfeuchtigkeit oder Temperatur besteht daraus, die entsprechenden Werte für die relative Luftfeuchtigkeit und der Temperatur für alle Bits vorher auszurechnen und in Form einer Lookup-Tabelle im Programm zu realisieren, damit der Mikrocontroller keine aufwändigen Rechnungen machen muss. Man bräuchte also bei einer Messung mit einer Genauigkeit von 14 Bit eine Lookup-Tabelle mit $2^{14} - 1 = 16383$ Einträgen. Um die Einträge zu reduzieren, haben wir die Genauigkeit der Messungen durch mehrfache Division durch Zwei auf acht Bit reduziert. Dafür braucht man nun eine Tabelle mit $2^8 - 1 = 255$ Einträgen. Realisiert wird das nun über das DB-Directive, was für die relative Luftfeuchtigkeit wie in Abbildung 2 aussehen kann.
Die erhaltenen Bits kann man jetzt als Index für den entsprechenden Wert der Bitfolge benutzen. Beispielsweise würde jetzt der Wert 1073, welcher mit einer Genauigkeit von 12 Bit gemessen wurde, durch 16 geteilt und als Index verwendet werden ($1073/16 = 67$). Der 67. Eintrag der Tabelle entspricht dem Wert 35% (zuvor 35.50%). Den Verlust der Genauigkeit nehmen wir hier für eine einfache Implementierung in Kauf. Der Zugriff auf den Eintrag erfolgt über den Befehl `lpm`. Dazu wird zunächst die Adresse der Tabelle um eine Stelle nach links verschoben in den Z Pointer geladen und auf die Adresse der Index addiert, um beim Verwenden von `lpm` den richtigen Eintrag zu laden.

```
LUT_Humidity:
.db 0,0,0,0,0,1,1,2,3,3,4,4,5,6,6,7
.db 7,8,8,9,10,10,11,11,12,12,13,14,14,15,15,16
.db 16,17,17,18,19,19,20,20,21,21,22,22,23,24,24,25
.db 25,26,26,27,27,28,28,29,30,30,31,31,32,32,33,33
.db 34,34,35,35,36,37,37,38,38,39,39,40,40,41,41,42
.db 42,43,43,44,44,45,45,46,46,47,47,48,49,49,50,50
.db 51,51,52,52,53,53,54,54,55,55,56,56,57,57,58,58
.db 59,59,60,60,61,61,62,62,63,63,64,64,64,65,65,66
.db 66,67,67,68,68,69,69,70,70,71,71,72,72,73,73,74
.db 74,75,75,75,76,76,77,77,78,78,79,79,80,80,81,81
.db 81,82,82,83,83,84,84,85,85,86,86,86,87,87,88,88
.db 89,89,90,90,90,91,91,92,92,93,93,93,94,94,95,95
.db 96,96,96,97,97,98,98,99,99,99,100,100,100,100,100,100
.db 100,100,100,100,100,100,100,100,100,100,100,100,100,100,100,100
.db 100,100,100,100,100,100,100,100,100,100,100,100,100,100,100,100
.db 100,100,100,100,100,100,100,100,100,100,100,100,100,100,100,100
```

Abbildung 2: Lookup-Tabelle für relative Luftfeuchtigkeit

4.6 Ausgabe über UART

Vor dem Benutzen der UART Schnittstelle muss man sie richtig initialisieren. Dazu schreibt man zunächst in das `UBRRH` und `UBRRH` Register einen Wert, der Abhängig von der Baudrate und der Frequenz des verwendeten Quarz' ist, wobei in das `UBRRH` Register die acht höherwertigen und in das `UBRRH` die acht niederwertigen Bits des Wertes stehen müssen. Der Wert wird für den asynchronen normalen Modus, welchen wir verwendet haben, mit folgender Formel berechnet:

$$UBRR = \frac{f_{OSC}}{16 * BAUD} - 1$$

Bei einer Frequenz von 4MHz und einer Baudrate von 9600 entspricht das dem Wert 25.
Zusätzlich muss man den Signalaufbau festlegen. Um einen 8N1 Aufbau zu haben, also einen Aufbau, welcher aus acht Datenbits, keine Paritätsbits und einem Stopbit besteht, muss man im `UCSRC` Register die Bits in den Feldern `URSEL`, `UCSZ1` und `UCSZ0` auf 1 und die Felder `UPM1`, `UPM0` und `USBS` auf 0 setzen. Dabei erlaubt `URSEL` das Schreiben in das `UCSRC` Register, `UCSZ1` und `UCSZ0` bestimmen die Anzahl der Datenbits, `UPM1` und `UPM0` bestimmen den Paritätsmodus und `USBS` die Anzahl der Stopbits. Standardmäßig sind alle Felder für unseren gewünschten Signalaufbau entsprechend gesetzt.
Um vom Mikrocontroller aus etwas Senden zu können, muss jetzt nur noch der Transmitter aktiviert werden. Dazu setzt man im `UCSRB` Register das `TXEN` Bit auf 1.
Um zu wissen, ob der Mikrocontroller bereit zum Senden ist, überprüft man das `UDRE` Bit im `UCSRA` Register. Ist das Bit gesetzt, bedeutet das, dass man ein Byte in das `UDR` Register laden und senden kann. Das `UDRE` Bit gibt hierbei lediglich an, ob das `UDR` Register leer und bereit für den Empfang von neuen Daten ist. Das Senden kann jetzt wie in Abbildung *** aussehen. ***hier bild einfügen***

Die gesendeten Werte werden als ASCII Werte interpretiert und dargestellt. Das bedeutet, dass man die Temperatur oder die relative Luftfeuchtigkeit in ASCII korrekt umwandeln muss, damit sie richtig dargestellt werden. Für dreistellige Zahlen benötigt man dafür drei Register. Möchte man beispielsweise die Zahl 123 übertragen, sendet man nacheinander die ASCII Symbole '1', '2', '3'. Für die Umwandlung verwenden wir eine Routine aus dem Internet, die binäre Zahlen in ASCII Werte umwandelt. Im Grunde zählt diese Routine die Hunderter, Zehner und Einer der Zahl und benutzt das Symbol '0' als Offset, um die Hunderter, Zehner und Einer jeweils als Symbol darzustellen.
Zu dem Umwandeln muss man noch beachten, dass auch negative Temperaturen richtig angezeigt werden muss. Da der Sensor eine maximale Temperatur von 123,8°C messen kann, also alle positiven Temperaturen mit sieben Bits darstellen kann, behandeln wir das MSB als Vorzeichen Bit, wobei die restlichen Bits in ASCII Symbole umgewandelt werden sollen. Anhand des MSB können wir nun '+' für positive und '-' für negative Temperaturen mit angeben.

4.7 Debug/Simulation

5 Schwierigkeiten

- Ground nicht angeschlossen
- Stromstecker vergessen
- Fehler ohne Debugger zu finden -> Lösungsansatz mit Oszilloskop oder an bestimmten Stellen vom Code LED zum leuchten bringen (um auch zu sehen ob man bestimmte Codeblöcke erreicht - in der richtigen Reihenfolge erreicht)
- AtmelStudio STK 500 hinzufügen - allgemein einen funktionierenden Programmer haben - Diamex dann auch mit richtigen Switches auf ON