Dokumentation des Sequenzers im Modul Programmiertechnik

Daniel Erich Raab und Justus Konstantin Schmidt

4. Juli 2018

Inhaltsverzeichnis

1 Vorwort

Dies ist die technische Dokumentation des Projekts im Modul Programmiertechnik von Daniel Raab und Konstantin Schmidt. Es wird der Aufbau der Software beschrieben und dargestellt, sowie die einzelnen Teilmodule. Das

Projekt wurde für das *MSP Education Board 3.0* der HTWK Leipzig entworfen und implementiert. Der Quellcode wurde in C mithilfe des *Code Composer Studio 6* geschrieben.

2 Struktur

2.1 Grundstruktur

Das Programm besitzt keinen sequentiellen Ablauf, da ein auf interrupts(engl. Unterbrechungen) basierender Ansatz gewählt wurde, um alle Funktionen gut implementieren zu können.

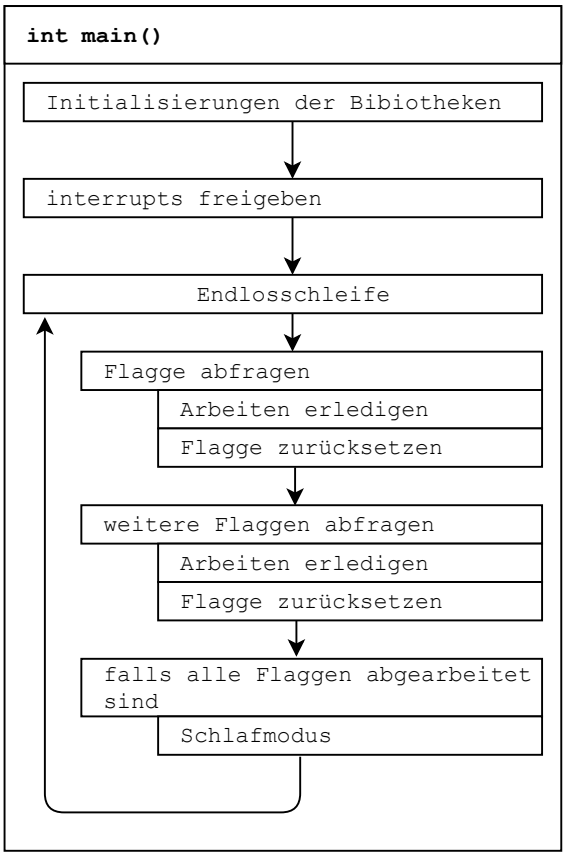

Abbildung 1: Der Grundablauf schematisch dargestellt

Der Grundablauf ist in Abbildung 1 zusehen. Es wird zuerst der Mikrorechner und alle benötigten Systeme initialisiert und dann in den interruptgesteuerten Betrieb übergegangen. Der Prozessor wird also in den Schlaf-

bzw. Low-Power-Modus versetzt und reagiert ab sofort nur noch auf auftretende interrupts. Die aufgerufene interrupt service routine (ISR) soll möglichst schnell abgearbeitet werden, also wird meist lediglich das interrupt ausgewertet und dann eine entsprechende Flagge gesetzt. Dann wird der Schlafmodus verlassen und die ISR beendet. Durch das Verlassen des Schlafmodus werden nun in der main()-Funktion die nächsten Befehle ausgeführt. Hier werden nun die Flaggen abgefragt und die entsprechenden Arbeiten erledigt. Sobald alle Flaggen abgefragt und abgearbeitet sind, wird der Prozessor wieder in den Schlafmodus versetzt. Diese Abarbeitung in der main()-Funktion hat den großen Vorteil, dass lang andauernde Arbeiten, wie z.B. Ansteuerung des Displays mit Wartezeiten, durch interrupts unterbrochen werden können, ohne das erst auf die lange andauernden Anweisungen gewartet werden muss. Diese Blockierung der schnell abzuarbeitenden ISRs passiert, wenn alle Aufgaben in ISRs passieren und nicht in der main()-Funktion. Zeitkritische Aufgaben z.B. Tastendrücke oder Töne könnten nicht richtig oder zeitnah ausgelesen bzw. ausgegeben werden. Eine Ausnahme dieser Regel betrifft die `play_tone`-Funktion der Tonbibliothek, siehe .

2.2 Modulstruktur

Die Programmierung des Mikrorechners erfolgte in Teilmodulen, die sich den verschiedenen Teilbereichen des Projekts widmen. Diese Teilmodule des Projekts wurden erst für sich entwickelt und sobald ein passabler Grad der Implementierung erreicht wurde in das Gesamtprojekt eingebunden und im Verbund getestet. Das Projekt wurde in folgende Module zerlegt:

- Sequenz-Datenstrukturen (*sequence.h*)
- Display-Ansteurung (*lcd.h*)
- Eingaben, also Taster und Drehgeber (*input.h*)
- Tonerzeugung (*tone.h*)
- LED-Matrix (*led_matrix.h*)

Die Steuerung des Programms passiert in der *main.c* über die Funktionen, die auf die Eingaben des Benutzers reagieren. Dort werden alle Bibliotheken und Module eingebunden und zum Projekt vereinigt.

3 Module

Hier werden die Module genauer in ihrem Aufbau und ihrer Struktur beschrieben. Die Modulnamen für die tatsächlichen Dateien sind in Klammern angegeben und umfassen die entsprechende *.h und *.c Dateien.

3.1 `main.c`

Die `main.c`-Datei ist der Einstiegspunkt des Programms und enthält die Funktion `int main()`. Es werden alle unserer Bibliotheken inkludiert, damit diese gleich zu Beginn durch ihre Initialisierungsfunktionen initialisiert werden können. Danach werden interrupts freigegeben und das Programm startet die Hauptschleife und damit den interrupt-basierten Ablauf, wie im Abschnitt Grundstruktur und in der Abbildung 1 erläutert.
Es werden außerdem die zwei ISRs der beiden Timer A und Timer B implementiert, da diese teilweise von mehreren Bibliotheken benötigt werden und deswegen von dieser zentralen Stelle diese Bibliotheken aufrufen. Dies wird für die Uhrzeit-, Ton- und LED-Matrix-Bibliotheken getan.
Es werden außer den ISRs noch ähnliche Funktionen definiert, die durch Bibliotheken für bestimmte Ereignisse aufgerufen werden. Dies sind `void new_minute()`, das von der Uhrzeit-Bibliothek aufgerufen wird, sobald eine neue Minute angebrochen ist und die Funktion `void alarm_ring()`, die signalisiert, dass ein Alarm ausgelöst worden ist und nun klingeln soll. Diese Funktionen wurden in die `main.c` Datei verlegt, da diese Ereignisse Veränderungen in mehreren Bibliotheken auslösen: `void new_minute()` bedeutet, dass die Alarme überprüft werden müssen und die LED-Matrix aktualisiert werden muss; `void alarm_ring()` bedeutet, dass das Menü in den Alarm-Modus übergehen muss und die Melodie des entsprechenden Alarms gestartet wird.

3.2 Display (*lcd*)

Die Display Bibliothek steuert das LC-Display und stellt verschiedene Methoden zur Beschreibung dessen zur Verfügung. Außerdem werden besondere Zeichen, hier eine Glocke, in das Display einprogrammiert und gespeichert. Das Display besitzt einen Cursor, der bestimmt wo neue Zeichen geschrieben werden.
Alle Funktionen des Display benötigen (relativ) viel Zeit, um mit dem Display-Controller zu kommunizieren und sollten demnach nicht in ISRs benutzt werden.

3.2.1 `void lcd_init()`

Das LC-Display wird gestartet und initialisiert über die Funktion `void lcd_init()`. Es wird dann die Initialisierungssequenz für das Display gesendet und es für die weitere Verwendung konfiguriert. Es wird das Display geleert und sowohl der Cursor als auch Blinken ausgeschaltet.

3.2.2 `void lcd_clear()`

Eine Funktion, die alles Angezeigte vom Display löscht und den Cursor zurück in die obere, linke Ecke (0, 0) setzt.

3.2.3 Schreibfunktionen

Als grundlegenden Schreibfunktionen, um das Display zu verwenden, sind

- `void lcd_write(unsigned char character)`,
- `void lcd_write_string(unsigned char string[])` und
- `void lcd_write_int(unsigned int number, int digits)`

für die verschiedenen wichtigen Datentypen implementiert. Es können einzelne Zeichen (`unsigned character`) gesendet werden, sowie ganze strings, also Felder von Zeichen, die jedoch mit dem terminierendem Zeichen `0x0` oder `'\0'` abgeschlossen werden müssen. Dies passiert automatisch, wenn ein string mit doppelten Anführungszeichen angegeben wird. Um Zahlen (`int`) anzuzeigen muss noch ein zweites Argument übergeben werden, das die Anzahl der Stellen bestimmt. Falls die Zahl weniger Stellen besitzt, wird der Rest mit Nullen aufgefüllt, also wird beispielsweise bei der Zahl 45 und drei Stellen `045` angezeigt.
All diese Funktionen schreiben immer ab der aktuellen Position des Cursors und der Cursor ist nach dem Schreiben beim nachfolgenden Zeichen des letzten neu auf das Display geschriebenen Zeichens.

3.2.4 `void lcd_set_cursor(unsigned char x, unsigned char y)`

Der Cursor kann mithilfe der Funktion `void lcd_set_cursor(unsigned char x, unsigned char y)` an eine beliebige Stelle im Display gesetzt werden. `x` steht für die Position innerhalb der Zeile (beginnend mit 0, also maximal 15) und `y` für eine der beiden Zeilen (0 oder 1).

3.2.5 Zeit und Datum

Es sind außerdem für dieses Projekt spezifische Funktionen implementiert, um eine Uhrzeit mit oder ohne Sekunden anzuzeigen (`void lcd_write_time(Time time, bool seconds)`), ein Datum mit oder ohne Wochentag anzuzeigen (`void lcd_write_date(Date date, bool weekdays)`) und das spezielle Zeichen Glocke auf das Display zu schreiben.

3.3 Eingabe (*input*)

In der Eingabe-Bibliothek werden die Taster sowie der Encoder ausgelesen und dem Rest des Programms in Form von Flaggen zur Auswertung zur Verfügung gestellt. Es werden die interrupts an Port 2 empfangen und ausgelesen, die jeweilige Flagge gesetzt und der Low-Power-Modus beendet, damit die Flaggen sofort ausgelesen und darauf reagiert werden kann.

3.3.1 `void button_init()`

Die Funktion initialisiert den Port 2 so, dass die interrupts ausgelöst und damit auf die Eingaben reagiert werden kann.

3.3.2 Flaggen

Es gibt für jeden Taster eine Flagge (`bool button_SW4` bis `bool button_SW1` entsprechend der Tasternummer) und zwei für die beiden Drehrichtungen des Encoders (`bool encoder_l` sowie `bool encoder_r`).

3.3.3 `__interrupt void P2_ISR()`

Die interrupt service routine für Port 2, die aufgerufen wird, sobald ein Taster oder der Encoder betätigt werden. Dort wird überprüft, ob der Encoder und wenn ja, in welche Richtung er gedreht wurde. Da die Taster prellen, dürfen die Tasterdrücke nicht sofort ausgewertet werden, da ansonsten bei jedem Tastendruck fälschlicherweise viele Tastendrücke erkannt werden würden. Deswegen wird über die Wartefunktion `void debounce_delay()` eine passende Wartezeit vor der Auswertung realisiert. Je nachdem welche Taster oder Encoder-Richtung erkannt wurde, wird die entsprechende Flagge gesetzt und der Prozessor aus dem Schlafmodus aufgeweckt, um in der `main()`-Funktion darauf zu reagieren.