TECHNISCHE HOCHSCHULE MITTELHESSEN

Fachbereich
Mathematik, Naturwissenschaften und Informatik

Praktikumsbericht

Mikroprozessortechnik
WS 15/16

Stand: 24.10.2015

Autoren:

Daniel Kreck
Mat.Nr.: 5012417
E-Mail: daniel.kreck@mni.thm.de

Leonard Schmischke
1234567
leonard.schmischke@mni.thm.de

Kursleitung: Prof. Dr. Klaus Wüst

I Kurzfassung

Lorem ipsum dolor sit amet, consetetur sadipscing elitr, sed diam nonumy eirmod tempor invidunt ut labore et dolore magna aliquyam erat, sed diam voluptua. At vero eos et accusam et justo duo dolores et ea rebum. Stet clita kasd gubergren, no sea takimata sanctus est Lorem ipsum dolor sit amet. Lorem ipsum dolor sit amet, consetetur sadipscing elitr, sed diam nonumy eirmod tempor invidunt ut labore et dolore magna aliquyam erat, sed diam voluptua. At vero eos et accusam et justo duo dolores et ea rebum. Stet clita kasd gubergren, no sea takimata sanctus est Lorem ipsum dolor sit amet.

Abstract

Das ganze auf Englisch.

II Inhaltsverzeichnis

1 Praktische Aufgabenstellungen

1.1 Endfassung Lauflicht

Diese Aufgabe verfolgte das Ziel, dass die Studierenden Erste Schritte mit den im Praktikum verwendeten Arbeitsmitteln machen konnten. Hierzu kam der Mikrocontroller MSP430 von Texas Instruments, ein Education Board des FTZ Leipzig, sowie die IAR Embedded Workbench IDE zum Einsatz.

Die Aufgabe bestand darin, ein C-Programm zu schreiben, dass die acht auf dem Praktikumsboard aufgebrachten Leuchtdioden endlos von rechts nach links und wieder zurück aufblinken lässt, sodass ein Lauflicht entsteht.

Zu beachten ist, dass die Logik invertiert ist. Das heißt ein Low-Pegel führt zum Leuchten der LED und ein High-Pegel zu deren erlöschen.

Bevor die LEDs angesprochen werden konnten, mussten die entsprechenden Leitungen als Ausgabeport geschaltet werden. Im Folgenden bestand unser weiteres Vorgehen darin eine Endlosschleife zu realisieren, in der geschaltet zwei weitere Iterationen vorgenommen wurden. In der einen wurde durch bitweises Verschieben nach links die Laufrichtung Rechts realisiert. In der anderen entsprechend umgekehrt. Hierzu wurde ein Bitmuster auf den Ausgabeport gelegt, dass dem Ein-/Ausschalten in der richtigen Reihenfolge der LEDs entspricht. Damit das menschliche Auge diese Operationen wahrnehmen kann, wurde durch eine weitere Schleife am Ende jedes Schaltvorgangs, eine Verzögerung umgesetzt.

```
1  /*****************************************************************
2
3     Datei blink1.c
4
5     Ablauffaehig auf Hardware: MSP430-Education-Board mit MSP430F2272
6
7     Laesst auf dem MSP430-Edu-Board eine LED blinken
8
9     Kommentar zur Schaltung auf Board:
10       P1.0 - P1.7              LED leuchtet wenn Ausgang=L (0)
11
12    Autor:    Daniel Kreck, Leonard Schmischke
13    Stand:    22.10.2015
14 */
15
16 #include <msp430x22x2.h>   \\Headerdatei mit Hardwaredefinitionen
17
18 // Funktions-Prototypen
19 void Warteschleife(unsigned long wartezeit);   // Software Warteschleife
```

```
20
21 int main(void) {
22
23   WDTCTL = WDTPW + WDTHOLD;     // Watchdog-Timer anhalten
24
25   // Hardware-Initialisierung
26   P1DIR = 0xFF;    // Port 1 arbeitet mit allen Leitungen (P1.0-P1.7) als
         Ausgabeport
27
28   unsigned char bitmaske = 0x01;
29   unsigned long wartezeit  = 25000;
30   while(1) {    // Endlosschleife
31     for (unsigned char i=0; i<7; i++) {    // hochzaehlen (von rechts
           nach links)
32       P1OUT = ~bitmakse;
33       bitmaske = bitmaske << 1;
34       Warteschleife(wartezeit);
35     }
36     for (unsigned char i=0; i<7; i++) {    // runterzaehlen (von links
           nach rechts)
37       P1OUT = ~bitmaske;
38       bitmaske = bitmaske >> 1;
39       Warteschleife(wartezeit);
40     }
41   }
42
43  // return 0;    // eigentlich richtig, Statement wird aber niemals
       erreicht
44 }
45
46 void Warteschleife(unsigned long wartezeit){
47   unsigned long i;
48   for (i=wartezeit; i>0; i--);    // Schleifenvariable herunterzaehlen
49 }
```

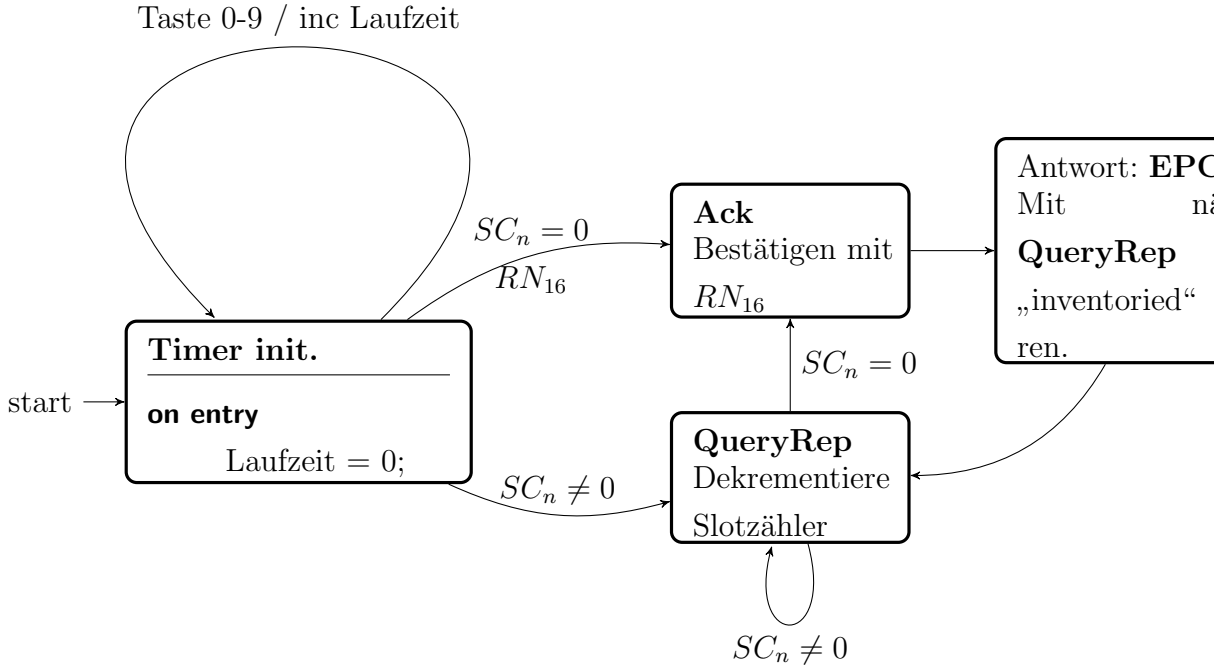
Listing 1: Lauflicht

1.2 Endfassung Taschenrechner

1.3 Endfassung Teatimer

1.4 Motorsteuerung

1.5 Fernbedienung



2 Analyse praktischer Probleme mit Interrupts

In der nachfolgenden Diskussion wird, durch die Analyse dreier Programme, auf häufig auftretende und in der Praxis zu beachtenden Tücken im Umgang mit der Interrupt-Technik aufmerksam gemacht.

2.1 Praktikumsaufgabe 22 - Interrupt-gesteuertes Programm I

Während der Analyse des Programms wurde beobachtet, dass der aktuelle Interrupt-Zählwert nicht korrekt auf dem LCD-Display angezeigt wird. Das Capture-/Compare-Register des Timers (TACCR0) wurde zur Problemeingrenzung mit verschiedenen Zählwerten belegt. Als Intervalle erwiesen sich 200-150-100-50 als praktikabel. Es konnte daraus geschlussfolgert werden, dass die ISR durch die beinhaltenden LCD-Operationen länger dauert als die Zeitdauer zwischen dem Auftreten der einzelnen Interrupts, wenn der Zählwert zu deren Auslösung zu klein gewählt ist. Einfacher formuliert, die Interrupts werden durch den niedrigen Zählwert so schnell ausgelöst, dass der vorherige Interrupt noch nicht abgearbeitet werden konnte. Dieses Verhalten lässt sich ungefähr bis zu einem Zählwert von

100 bis 150 beobachten. Darüber hinaus werden vorherige ISRs vor dem Eintreffen des nächsten Interrupts vollständig durchgeführt und die Anzahl Interrupts korrekt gezählt und auf dem Display angezeigt.

Es gibt mehrere Ansätze diesem Sachverhalt zu begegnen:

1. **Lösungsvorschlag:** Unter der Prämisse, dass die langwierigen LCD-Operationen in der ISR stehenbleiben, könnte man diese nur alle x-Aufrufe ausführen. Das Display wird demnach intervallartig aktualisiert.

   Pro: regelmäßge Darstellung des aktuellen Wertes auf dem Display, auch bei kleinen CC0-Zählwerten.

   Contra: verfälscht das Messergebnis leicht.

2. **Lösungsvorschlag:** In der ISR wird lediglich die Zähler-Variable inkrementiert. Die LCD-Operationen werden in die Endlosschleife innerhalb der main-Funktion verschoben, damit die einzelnen ISRs schneller abgearbeitet werden können.

   Pro: verfälscht das Messergebnis nicht.

   Contra: nur bei größeren Zählwerten möglich, da bei kleinen keine Zeit zwischen den ISR-Aufrufen vorhanden ist um in die main-Funktion zurückzukehren. Auf eine ISR folgt direkt die Nächste.

2.2 Praktikumsaufgabe 23 - Interrupt-gesteuertes Programm II

Während der Analyse des Programms wurde beobachtet, dass die LEDs und das LCD-Display einen unterschiedlichen Wert anzeigen, obwohl sie nacheinander über die gleiche Variable beschrieben werden. Diese Variable „Zaehler“ wird jedes mal in der zugehörigen ISR inkrementiert wenn ein Interrupt von Port 2 ausgelöst wird. Am Ende dieser ISR wechselt ein Flag namens „Zaehlerveraendert“ auf TRUE. Infolge dessen wird in der Endlosschleife die Aktualisierung erst des LCDs, dann der LEDs vorgenommen.

Das Problem bei dieser Art von Programmierung ist, dass nicht bedacht wurde, dass zwischen diesen beiden Aktualisierungen ein erneuter Interrupt ausgelöst werden kann, der die Variable „Zaehler“ verändert. Das Resultat ist, dass das LCD mit einem anderen Wert beschrieben wurde, als es nachfolgend die LEDs werden, wodurch die Darstellung inkonsistent ist.

Diese Problematik ist als Shared-Data-Bug bekannt. Ein Ansatz diesem in dieser Situation zu begegnen, ist es eine Hilfsvariable anzulegen, die am Anfang der Endlosschleife mit

dem Wert von „Zaehler“ beschrieben wird. Anschließend wird diese Variable zur Aktualisierung der beiden Anzeigegeräte verwendet, wodurch sich zwischenzeitliche Interrupts nicht bemerkbar machen. Die Wertzuweisung an die Hilfsvariable kann ebenfalls nicht von einem Interrupt unterbrochen werden, da diese vom Typ Character wäre und somit innerhalb eines Taktes mithilfe eines Maschinenbefehls vonstatten ginge.

2.3 Praktikumsaufgabe 24 - Interrupt-gesteuertes Programm III

Während der Analyse des Programms wurde beobachtet, dass die zweite If-Bedingung der Endlosschleife nach einer gewissen Zeit erfüllt und somit der Text „Seltsam, oder?!“ auf das LCD-Display ausgegeben wurde. Dies dürfte eigentlich nie passieren, da die Bedingung so konstruiert ist, dass sie durch den bestehenden Code nicht erfüllt werden kann.

Das Problem ist hier der Datentyp Long in Verbindung mit der Interrupt-Technik. Der MSP430 ist ein 16-Bit-Prozessor. Dies hat zur Folge, dass ein Integer-Typ typischerweise 16 Bit und ein Long-Typ 32 Bit groß ist. Wenn eine Variable eines solchen Typs bspw. einen Wert zugewiesen bekommt, sind zwei Maschinenbefehle die Folge. Der eine überträgt das Low-Word und der andere das High-Word, denn die Variable muss auf zwei 16-Bit Speicherplätze verteilt werden.

Zur Veranschaulichung der Problematik betrachte man untenstehenden Assemblercode, der aus diesem C-Code hervorgeht:

```
1 if ( (Zaehler > (oldZaehler+100) ) || (Zaehler < (oldZaehler-100) ) )
      {...}
```

Listing 2: C-Code MSP430

Der Wert von OldZaehler wird in Register 14 und 15 der CPU geladen. Zuerst das Low-Word (0x204) und dann das High-Word (0x206).

```
1 008070    421E 0204      mov.w   //&oldZaehler,R14
2 008074    421F 0206      mov.w   //&0x206,R15
```

Listing 3: Assemblercode MSP430

Auf das Low-Word wird nun die Dezimalzahl 100 (0x64) addiert und das Carry-Bit (Übertrag) auf das High-Word geschrieben, womit dessen vorheriger Inhalt hinfällig ist.

```
1 008078    503E 0064      add.w   //0x64,R14
2 00807C    630F           adc.w   //R15
```

Listing 4: Assemblercode MSP430

Danach werden die Werte in zwei Stufen verglichen, gegebenfalls gesprungen und ein äquivalentes Prozedere mit dem zweiten Teil der If-Anweisung durchgeführt:

```
1  00807E   912F 0202    cmp.w   //&0x202,R15
2  008082   3812         jl      //0x80A8
3  008084   2003         jne     //0x808C
4  008086   921E 0200    cmp.w   //&Zaehler,R14
5  00808A   280E         jnc     //0x80A8
6
7  00808C   421E 0204    mov.w   //&oldZaehler,R14
8  008090   421F 0206    mov.w   //&0x206,R15
9  008094   503E FF9C    add.w   //#0xFF9C,R14
10 008098   633F         addc.w  //#0xFFFF,R15
11 00809A   9F82 0202    cmp.w   //R15,&0x202
12 00809E   3804         jl      //0x80A8
13 0080A0   23D8         jne     //0x8052
14 0080A2   9E82 0200    cmp.w   //R14,&Zaehler
15 0080A6   2FD5         jc      //0x8052
```

Listing 5: Assemblercode MSP430

Wird nun während des zweistufigen Vergleichs ein Interrupt ausgelöst und die Variable Zaehler durch die ISR inkrementiert, ist die If-Bedingung erfüllt.

Lösungsansatz: Dieses Problem lässt sich bei dem Datentyp Long nicht beheben. Man kann ihn lediglich versuchen so gut es geht zu vermeiden oder in solchen kritischen Abschnitten Interrupts kurzfristig abschalten. Zumindest in soweit, dass Variablen der kritischen Abschnitte nicht durch ISRs verändert werden, während man sich gerade in einem solchen Abschnitt befindet.

3 Theoretische Aufgabenstellungen

Unter diesem Abschnitt sind die, zu bestimmten Aufgaben in der Praktikumsanleitung vorkommenden Frageblöcke, mit einer ausführlichen Antwort versehen worden.

3.1 Block 1 - Erste Schritte / IDE

Was ist ein Crosscompiler? Ein Crosscompiler erzeugt beim Übersetzungsvorgang einen Zielcode der für eine andere Prozessorarchitektur als die Eigene ausgelegt ist. Bei uns ist dieser Teil der IAR-Entwicklungsumgebung und erzeugt Zielcode für den MSP430, anstatt für den in der Workstation verbauten Prozessor.

Was ist eine JTAG-Schnittstelle? Eine JTAG-Schnittstelle wird unter anderem für die Übertragung des Maschinencodes vom Entwicklungsrechner zum Education Board benutzt. Des Weiteren kann mit ihr die Programmausführung gesteuert werden, was bedeutet das In-System-Debugging möglich ist.

3.2 Block 2 - Erste Schritte / Blinkprogramm

Warum kann der Parameter Anzahl in der Funktion Warteschleife nicht vom Typ int sein? Bei unserem MSP430 handelt es sich um ein 16-Bit-Prozessorsystem, wo ein Integer typischerweise dann auch 2 Byte groß ist. Ein signed Integer hat also einen Datenbereich von -32.768 bis 32.767. Die ursprüngliche Schleife war auf einen Wert von 50.000 notiert, weswegen ein Überlauf stattgefunden hätte. Das führt letztlich dazu, dass die Laufbedingung der For-Schleife i>0 nicht zutrifft, die Schleife nie läuft und somit die Warteschleife nichtig wird.

Welche Hardwaregruppen werden angesprochen? Das Kontrollregister des Watch-Dog-Timers WDTCTL, das Richtungsregister P1DIR und das Ausgangspufferregister P1OUT werden angesprochen.

Warum muss man keine Hardwareadressen angeben um die Hardware anzusprechen? In der inkludierten Headerdatei msp430x22x2.h sind für die Hardwareadressen symbolische Namen, mittels Makrodefinitionen, angelegt.

Wie ist die Hardwareadresse von P1OUT? Wie kann man das herausfinden? Die Hardwareadresse von P1OUT ist 0x0021. Diese Information kann man entweder in der Headerdatei msp430x22x2.h oder unter der Rubrik 8.3 Digital I/O Resgisters im Family Users Guide (S.333) des Mikrocontrollers nachlesen.

Warum wird der Watchdog Timer ausgeschaltet? Der Watch-Dog-Timer ist standardmäßig eingeschaltet und hat die Aufgabe ein Versagen des Systems durch Softwarefehler zu vermeiden. Die Software muss typischerweise dem Watch-Dog-Timer in regelmäßigen Abständen mitteilen, dass sie sich noch in einem konsistenten Zustand befindet. Erfolgt dies nicht, setzt der Watchdog das Gerät automatisch in einen definierten Ausgangszustand zurück. Da wir in der ersten Praktikumsaufgabe LED-Lauflicht softwaretechnisch eine Endlosschleife umgesetzt haben musste dieser einerseits zuvor ausgeschaltet werden. Andererseits handelt es sich bei unseren Praktikumsaufgaben nicht um kritische Anwendungen, wo ein aktiver Watch-Dog-Timer vonnöten wäre.

Wie funktioniert In-System-Debugging? Nach der Übersetzung des C-Programmes wird dieses über die JTAG-Schnittstelle in den Flash-Speicher des Mikrocontrollers geladen. Anschließend wird es zur Ausführung gebracht. Dabei stehen zusätzlich verschiedene Debug-Möglichkeiten wie das setzen von Breakpoints, Inspektion der Speicherplätze und Register, Stepping, Tracing usw. zur Verfügung. Das Debugging erfolgt echt auf dem Chip, sodass die Befehle direkt von der Hardware umgesetzt werden. Dazu wird eine spezielle Kontrollhardware benötigt, die die Kontrolle über den Programmlauf behält und Kommandos und Informationen über die JTAG-Schnittstelle mit der IDE synchronisiert.

3.3 Block 3 - Interrupt über Port 2

Was bedeutet die Zeile "#pragma vector=PORT2_Vector"? Dieser Funktionsvorsatz ist eine Anweisung an den Compiler, die Adresse dieser Funktion an die Stelle die für Port 2 in der Interrupt-Vektoren-Tabelle vorgesehen ist, einzutragen. Wenn nachfolgend ein Interrupt von Port 2 ausgelöst wird, wird dort über die hinterlegte Adresse genau diese Funktion (ISR) aufgerufen und ausgeführt.

Warum wird hier vor den Funktionsnamen "__interrupt"gesetzt? Es handelt sich um einen ergänzenden Typbezeichner. Dieser weist den Compiler an, die damit versehene Funktion als Interrupt-Service-Routine zu übersetzen, also speziellen Programmcode zusätzlich zu erzeugen. Dieser sorgt insbesondere dafür, dass alle Register und Flags vor dem Aufruf (auf dem Stack) gesichert und nach dem Aufruf zurückgeschrieben werden, damit unter anderem an die vorherige Stelle im Programmcode zurückgesprungen werden kann und die zuvor benutzen Register und Flags auch die zu erwartenden Werte aufweisen.

Warum können keine weiteren Interrupts eintreten, wenn man in der Interrupt–Service-Routine ist? Diese Aussage ist nicht korrekt. Es ist sowohl möglich, dass zwei oder mehrere Interrupts gleichzeit ausgelöst werden als auch, dass während eine ISR bearbeitet wird ein weiterer Interrupt eintritt. Beide Situationen werden durch eine Prioritätenregelung aufgelöst. So wie das laufende Programm durch einen Interrupt unterbrochen werden kann, kann ein Interrupt niedrigerer Priorität durch einen Interrupt höherer Priorität unterbrochen werden.

3.4 Block 4 - Zyklische Interrupts durch den Timer

Warum läuft der Timer A nach dem Befehl "TACTL = TASSEL0 + TACLR;"noch nicht gleich? Der Befehl bewirkt lediglich, dass das Controlregister des Timers A als Eingangstakt TAClock zugeführt und das Zählregister auf 0 zurückgesetzt wird. Der Timer

befindet sich allerdings noch im Stop-Mode. Über die Mode-Control-Bits kann dieser in verschiedenen Modi gestartet werden. Dies wird erreicht, in dem bspw. die Makrodefinition MC_1 - MC_3 gleich mit verodert werden (TACTL = TASSEL0 + TACLR + MC_1), um den Timer direkt starten zu lassen oder dem Anwendungszweck entsprechend, erst später (TACTL = MC_1).

Welche Signale kann man dem Eingang des Timers (außer AClock) zuführen? Als interne Taktquelle kann außer der AClock die SMClock verwendet werden. Als externe Taktquellen können TAClock und INClock zugeführt werden.

Wie funktioniert der Up Mode des Timers A und welche Modes gibt es noch? Im Up-Mode zählt der Timer aufwärts, bis er den zuvor spezifizierten Wert im Capture-/Compare-Register erreicht. Beim nächsten Takt beginnt er erneut von 0x0000 zum angeben Wert hochzuzählen. Dieser kann maximal 0xFFFF sein, welches äquivalent zum Continuous-Mode wäre. Den Up-Mode kann man als Zählbetrieb des Timers bezeichnen, bei dem von außen kommende Impulse mithilfe der Interrupt-Technik gezählt werden können. Wenn diese eingeschaltet werden, wird jedes mal bei Erreichen des Zählwertes vor dem Umschalten auf 0x0000 ein Interrupt ausgelöst. Zu den bereits genannten Stop-, Up- und Continuous-Mode gibt es noch den Up/Down-Mode, in dem bei Erreichen des Caputre-/Compare-Wertes von diesem an bis 0x0000 runtergezählt wird und anschließend wieder hoch, anstatt wie beim Up-Mode direkt auf 0x0000 zu wechseln.

Mit welcher Taktfrequenz läuft unser Edu-Board? Das Edu-Board läuft mit dem Takt, mit dem die CPU betrieben wird. Diese liegt zwischen 12 kHz und 16 MHz.

3.5 Block 5 - Impulse von rotierender Welle zählen und anzeigen

Aus welcher Quelle sollten die Impulse am Zählereingang kommen? Sie sollten von TAClock bzw. TBBlock kommen, je nachdem welchen Timer man zum Impulszählen benutzt.

Welchen Timer könnte man benutzen? Es kann sowohl Timer A als auch Timer B benutzt werden. Es sollte allerdings beachtet werden, dass Timer A ein festes 16-Bit-Zählregister hat, wohingegen man dieses bei Timer B softwaremäßig auf 8, 10, 12 oder eben 16 Bit einstellen kann. Weitere kleine Unterschiede können unter 13.1.1 Similarities and Differences From Timer_A des Family Users Guide nachgelesen werden.

Welche Impulsquelle wird benutzt? Die rotierende Scheibe ist schwarz/weiß gestreift, wodurch es einem angebrachten Helligkeitssensor möglich ist bei Farbwechsel die Impulse auszulösen.

In welchem Mode sollte der Timer betrieben werden? Der Timer sollte im Continuous-Mode betrieben werden, da es einfach nur gilt die von außen kommenden Impulse zu zählen und um möglichst selten Überläufe mit einrechnen zu müssen der maximal Wert 0xFFFF sinnvoll wäre.

Wo nimmt man die nötigen Einstellungen vor? Sie werden im Controlregister des entsprechenden Timers vorgenommen. Bei Timer A wäre dies TACTL und bei Timer B TBCTL.

Wie oft sollte man die Anzeige aktualisieren? Es sollte ein Wert gewählt werden, der die langsamen LCD-Operationen berücksichtigt, damit keine Anzeigefehler auftreten. Da das Display lediglich vom menschlichen Auge abgelesen wird, reicht ein Intervall von einer halben bis einer Sekunde aus.

Kann bei Zählerbetrieb ein Low-Power-Mode genutzt werden, wenn ja, welche? Es können die Low-Power-Modi 0 bis 3 genutzt werden. Der Low Power Mode 4 kann nicht zum Einsatz kommen, da dort auch alle Oszillatoren gestoppt werden, durch die der Timer versorgt wird.

3.6 Block 6 - Anzeige der Drehzahl des rotierenden Rades

Wie kann grundsätzlich eine Drehzahl gemessen werden? Die Drezahl gibt die Anzahl der Umdrehungen N pro Zeitintervall T an, also wie häufig eine Umdrehung in dieser Zeit vollzogen werken konnte. Mathematisch ausgedrückt:

$$n = \frac{\Delta N}{\Delta T}$$

In unserem Fall des Helligkeitssensors macht es Sinn, die Impulse pro Sekunde zu messen, zu ermitteln wie viele Impulse eine vollständige Umdrehung ausmachen und die Drehzahl in der Einheit Minute auszugeben.

Achtung Praxis: Überläufe des Zählregisters beachten und vorzugsweise mit gekürzten Werten rechnen.

Wie viele Timer braucht man? Man braucht 2 Timer. Der eine misst die Impulse, die zur Berechnung der Umdrehungsanzahl benötigt werden (im Continous-Mode). Der andere gibt das Zeitintervall an, indem die Drezahlberechnung dann zyklisch stattfindet. Dazu wird der Up-Mode benutzt, das Capture-/Compare-Register mit einem entsprechenden Wert belegt, Interrupts eingeschaltet und bei jeder Auslösung eines solchen die Drehzahlberechnung durchgeführt.

3.7 Block 7 - Analoges Signal von Potentiometern anzeigen

Wozu werden die beiden internen ADC-Kanäle genutzt? Sie werden für den internen Temperatur-Sensor und zur Größenermittlung der externen Referenzspannung genutzt.

Welche Konvertierungsarten (Conversion Modes) kennt der ADC10? Es gibt vier Konvertierungsarten:

- single channel single-conversion
- sequence-of-channels
- repeat single channel
- repeat sequence-of-channels

3.8 Block 8 - Interrupt über Port 2 unter Berücksichtigung energieeffizenter Programmierung

Bei Eintritt eines Interrupts beendet der MSP430 automatisch den Low-Power–Mode und wechselt in den Active-Mode, Warum ist das sinnvoll und wie wird es technisch durchgeführt? Diese Vorgehen ist nicht nur sinnvoll, sondern auch notwendig, weil der CPU-Takt unterbrochen wurde und somit keine Maschinenbefehle ausgeführt werden können, um dies manuell zu bewerkstelligen. Technisch umgesetzt wird dies über das Statusregister der CPU. In diesem kann das Bit CPUOFF gesetzt (1: CPU inaktiv) und gelöscht werden (0: CPU aktiv). Standardmäßig steht dieses auf 0. Wird die CPU verleitet einen Low-Power-Mode zu schalten, wird das Bit auf 1 gesetzt. Kommt während sich die CPU im Low-Power-Mode befindet ein Interrupt, wird dieses Bit gelöscht, die ISR abgearbeitet und danach das Bit erneut gesetzt, wodurch das System zurück in den Low-Power-Mode versetzt wird.

Welche Low-Power-Modes sind bei dem oben stehenden Programm möglich? Alle Low-Power-Modes (0 bis 4) sind möglich, da Interrupts nicht durch diese abgeschaltet

werden.

Welche Low-Power-Modes sind möglich beim Taschenrechner? Alle Low-Power-Modes (0 bis 4) sind möglich.

Welche Low-Power-Modes sind möglich beim Tea-Timer? Es können die Low-Power-Modi 0 bis 3 genutzt werden. Der Low Power Mode 4 kann nicht zum Einsatz kommen, da dort auch alle Oszillatoren gestoppt werden, durch die der Timer versorgt wird. Die Zeit könnte nicht mehr runtergezählt werden.

Kann man an unserem Education System das Absinken des Versorgungsstromes beim Eintritt in einen Low-Power-Mode messen? Ja kann man.

4 Aufgabenstellungen des MPT-Skripts

5 Einleitung

Dieses Kapitel enthält Beispiele zum Einfügen von Abbildungen, Tabellen, etc.

Wie man wissenschaftliche Arbeiten mit Latex schreibt, findet man u.a. in [Sch11]

5.1 Bilder

Zum Einfügen eines Bildes, siehe Abbildung 1, wird die *minipage*-Umgebung genutzt, da die Bilder so gut positioniert werden können.

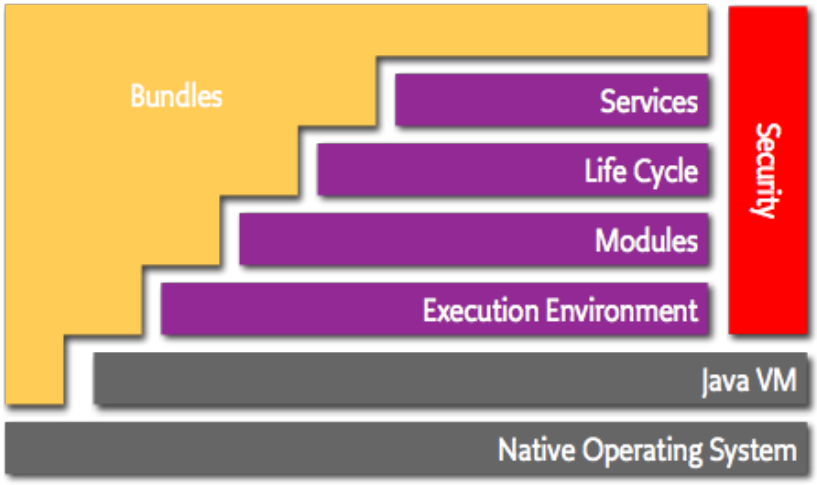


Abbildung 1: OSGi Architektur[1]

5.2 Tabellen

In diesem Abschnitt wird eine Tabelle (siehe Tabelle 1) dargestellt.

Name	Name	Name
1	2	3
4	5	6
7	8	9

Tabelle 1: Beispieltabelle

[1] Quelle: `http://www.osgi.org/Technology/WhatIsOSGi`

5.3 Auflistung

Für Auflistungen wird die *compactitem*-Umgebung genutzt, wodurch der Zeilenabstand zwischen den Punkten verringert wird.

- Nur
- ein
- Beispiel.

5.4 Listings

Zuletzt ein Beispiel für ein Listing, in dem Quellcode eingebunden werden kann, siehe Listing 6.

```
1  int ledPin = 13;
2  void setup() {
3      pinMode(ledPin, OUTPUT);
4  }
5  void loop() {
6      digitalWrite(ledPin, HIGH);
7      delay(500);
8      digitalWrite(ledPin, LOW);
9      delay(500);
10 }
```

Listing 6: Arduino Beispielprogramm

Oder der Quellcode ist in einem gesonderten File:

```
1  class Hello{
2
3  public:
4
5    void sayHello(){
6      cout<< "Hello";
7    }
8  }
```

Listing 7: Hello.cpp

5.5 Tipps

Die Quellen befinden sich in der Datei *literatur.bib*. Ein Buch- und eine Online-Quelle sind beispielhaft eingefügt. [Vgl. [Mus13], [SK99]]

Abkürzungen lassen sich natürlich auch nutzen (Open Service Gateway initiative (OSGi)). Weiter oben im Latex-Code findet sich das Verzeichnis. Vgl.: Open Gateway for Energy Management (OGEMA)

6 Tabellen online erstellen

Unter `http://www.tablesgenerator.com` findet man einen Tabellengenerator. Tabelle 2 wurde damit erstellt.

a	b	c	d	e
f	g	h	i	j
k	l	m	n	o
p	q	r	s	t
u	v	w	x	y
z	1	2	3	4
5	6	7	8	9

Tabelle 2: Beispieltabelle

7 Kapitel

Lorem ipsum dolor sit amet.

7.1 Unterkapitel

Lorem ipsum dolor sit amet, consetetur sadipscing elitr, sed diam nonumy eirmod tempor invidunt ut labore et dolore magna aliquyam erat, sed diam voluptua. At vero eos et accusam et justo duo dolores et ea rebum. Stet clita kasd gubergren, no sea takimata sanctus est Lorem ipsum dolor sit amet. Lorem ipsum dolor sit amet, consetetur sadipscing elitr, sed diam nonumy eirmod tempor invidunt ut labore et dolore magna aliquyam erat, sed diam voluptua. At vero eos et accusam et justo duo dolores et ea rebum. Stet clita kasd gubergren, no sea takimata sanctus est Lorem ipsum dolor sit amet.

7.2 Unterkapitel

Lorem ipsum dolor sit amet, consetetur sadipscing elitr, sed diam nonumy eirmod tempor invidunt ut labore et dolore magna aliquyam erat, sed diam voluptua. At vero eos et accusam et justo duo dolores et ea rebum. Stet clita kasd gubergren, no sea takimata sanctus est Lorem ipsum dolor sit amet. Lorem ipsum dolor sit amet, consetetur sadipscing elitr, sed diam nonumy eirmod tempor invidunt ut labore et dolore magna aliquyam erat, sed diam voluptua. At vero eos et accusam et justo duo dolores et ea rebum. Stet clita kasd gubergren, no sea takimata sanctus est Lorem ipsum dolor sit amet.

8 Kapitel

Lorem ipsum dolor sit amet.

8.1 Unterkapitel

Lorem ipsum dolor sit amet, consetetur sadipscing elitr, sed diam nonumy eirmod tempor invidunt ut labore et dolore magna aliquyam erat, sed diam voluptua. At vero eos et accusam et justo duo dolores et ea rebum. Stet clita kasd gubergren, no sea takimata sanctus est Lorem ipsum dolor sit amet. Lorem ipsum dolor sit amet, consetetur sadipscing elitr, sed diam nonumy eirmod tempor invidunt ut labore et dolore magna aliquyam erat, sed diam voluptua. At vero eos et accusam et justo duo dolores et ea rebum. Stet clita kasd gubergren, no sea takimata sanctus est Lorem ipsum dolor sit amet.

8.2 Unterkapitel

Lorem ipsum dolor sit amet, consetetur sadipscing elitr, sed diam nonumy eirmod tempor invidunt ut labore et dolore magna aliquyam erat, sed diam voluptua. At vero eos et accusam et justo duo dolores et ea rebum. Stet clita kasd gubergren, no sea takimata sanctus est Lorem ipsum dolor sit amet. Lorem ipsum dolor sit amet, consetetur sadipscing elitr, sed diam nonumy eirmod tempor invidunt ut labore et dolore magna aliquyam erat, sed diam voluptua. At vero eos et accusam et justo duo dolores et ea rebum. Stet clita kasd gubergren, no sea takimata sanctus est Lorem ipsum dolor sit amet.

9 Kapitel

Lorem ipsum dolor sit amet.

9.1 Unterkapitel

Lorem ipsum dolor sit amet, consetetur sadipscing elitr, sed diam nonumy eirmod tempor invidunt ut labore et dolore magna aliquyam erat, sed diam voluptua. At vero eos et accusam et justo duo dolores et ea rebum. Stet clita kasd gubergren, no sea takimata sanctus est Lorem ipsum dolor sit amet. Lorem ipsum dolor sit amet, consetetur sadipscing elitr, sed diam nonumy eirmod tempor invidunt ut labore et dolore magna aliquyam erat, sed diam voluptua. At vero eos et accusam et justo duo dolores et ea rebum. Stet clita kasd gubergren, no sea takimata sanctus est Lorem ipsum dolor sit amet.

9.2 Unterkapitel

Lorem ipsum dolor sit amet, consetetur sadipscing elitr, sed diam nonumy eirmod tempor invidunt ut labore et dolore magna aliquyam erat, sed diam voluptua. At vero eos et accusam et justo duo dolores et ea rebum. Stet clita kasd gubergren, no sea takimata sanctus est Lorem ipsum dolor sit amet. Lorem ipsum dolor sit amet, consetetur sadipscing elitr, sed diam nonumy eirmod tempor invidunt ut labore et dolore magna aliquyam erat, sed diam voluptua. At vero eos et accusam et justo duo dolores et ea rebum. Stet clita kasd gubergren, no sea takimata sanctus est Lorem ipsum dolor sit amet.

Vgl. [FLF+08], [Bie08] und [Bie05]

10 Quellenverzeichnis

[Bie05] Bienhaus, Diethelm: A Pattern Language for the Network of Things. In: *Proceedings of 10th European Conference on Pattern Languages of Programs (EuroPlop 2005)*, 2005

[Bie08] Bienhaus, Diethelm: Patterns for Managing Data in Complex Automatic Identification and Data Capturing Environments. In: Schümmer, Till (Hrsg.): *EuroPLoP* Bd. 610, CEUR-WS.org, 2008 (CEUR Workshop Proceedings)

[FLF+08] Floerkemeier, Christian (Hrsg.) ; Langheinrich, Marc (Hrsg.) ; Fleisch, Elgar (Hrsg.) ; Mattern, Friedemann (Hrsg.) ; Sarma, Sanjay E. (Hrsg.): *The Internet of Things. First International Conference, IOT 2008, Zurich, Switzerland, March 26-28, 2008, Proceedings.* Bd. *4952.* Berlin Heidelberg New York : Springer, März 2008 (LNCS). – Available in print and online.

[Mus13] Mustermann, Max: *Ich bin ein Buch.* Verlag, 2013

[Sch11] Schlosser, Joachim: *Wissenschaftliche Arbeiten schreiben mit LATEX : Leitfaden für Einsteiger; [schnell zur fertig gesetzten Arbeit - ohne Vorkenntnisse; lösungsorientierte und verständliche Beschreibungen; von Tabellen und Formeln über Grafiken bis zum Literaturverzeichnis].* Heidelberg; München; Landsberg; Frechen; Hamburg : mitp, 2011. – ISBN 9783826691027 3826691024

[SK99] Stallman, Richard M. ; Krempl, Stefan: *Software muß frei sein!* Website, 1999. – Online erhältlich unter `http://www.heise.de/tp/deutsch/inhalt/te/2860/1.html`; abgerufen am 8. Januar 2005.

Anhang

A GUI

Ein toller Anhang.

Screenshot

Unterkategorie, die nicht im Inhaltsverzeichnis auftaucht.

B Abbildungsverzeichnis

C Tabellenverzeichnis

D Listing-Verzeichnis

E Abkürzungsverzeichnis

OSGi Open Service Gateway initiative

OGEMA Open Gateway for Energy Management

Erklärung

Hiermit versichere ich, dass ich meine Abschlussarbeit selbständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt habe.

Datum: .....

(Unterschrift)