MIKROPROZESSORPRAKTIKUM

WS2019

Einführung

Einfuehrung zum Mikroprozessorpraktikum

Name, Vorname	Matrikelnummer	Anmerkungen
Datum	Raster (z.B. Mi3x)	Testat/Datum

Legende: V:Vorbereitung, D: Durchführung, P: Protokoll/Dokumentation, T: Testat

Inhaltsverzeichnis

Laborordnung

Sinn dieser Laborordnung ist die Festlegung von Regeln für die Benutzung der Labore in D10

Jeder ordentliche Student des Fachbereich wird in den Grundlagenveranstaltungen auf die gültige Laborordnung hingewiesen.

1. Der Notausschalter des Labors D10/0.32 befindet sich über dem Lichtschalter. Der Notausschalter ist im Notfall zu betätigen, um sämtliche elektrische Geräte stromlos zu schalten. Achten Sie bei der Einführung auf die entsprechenden Hinweise.
2. Im Labor darf maximal Paarweise an den Geräten gearbeitet werden. Das Labor ist für 8 Gruppen ausgelegt.
3. Es ist nicht gestattet sich alleine im Labor aufzuhalten.
4. Die Ersthelfer sind Herr Rudi Scheitler (Raum 0.33 / Tel.: 38465), Herr Manfred Pester (Raum 0.33 / Tel.: 38428), Herr Sergio Vergata (Raum 0.37 / Tel.: 38491) und Frau Bettina Kurz (Raum 0.36 / Tel.:38453). Wenden Sie sich bitte im Falle einer Verletzung direkt an eine dieser Personen . Das Erste-Hilfe-Material befindet sich im Eingangsbereich Südseite.
5. Es ist nicht gestattet Kabel zu entfernen, Gehäuse zu öffnen und Hardware (außer USB-Sticks) zu installieren oder Änderungen an der Laborinfrastruktur vorzunehmen. Sollte etwas nicht funktionieren, oder es wird etwas benötigt, welches die vorhandene Infrastruktur nicht abdeckt, so wenden Sie sich an den Betreuer des Labors oder direkt an den zuständigen Laboringenieur Manfred Pester (Raum 0.33 / Tel.: 38428).
6. Fahren Sie die von Ihnen benutzten Geräte am Ende Ihres Praktikum/Ihrer Übung herunter und schalten diese aus, es sei denn Sie bekommen vom zuständigen Betreuer andere Anweisungen.
7. Speisen und Getränke sind an den Arbeitsplätzen im Labor nicht erlaubt.
8. Bei der Benutzung des Labordrucker ist Sorgfalt und Sparsamkeit oberstes Gebot.
9. Evtl. ausgestellte Dokumentationen dienen der Laborarbeit und müssen im Raum verbleiben.
10. Die Benutzung von Mobiltelefonen ist untersagt. Schalten Sie vor dem Betreten des Raumes die Geräte ab (Flugzeugmodus ist ok). In dringenden Fällen können Sie sich über das Labortelefon mit der Nummer 06151 168433 anrufen lassen.
11. Hängen Sie Ihre Kleidung (Mäntel, Jacken, ..) an die dafür vorgesehenen Kleiderständer und nicht über die Stühle.
12. Deponieren Sie Taschen, Laptops u.s.w. nicht in den Gängen, sondern möglichst an den Seiten des Labors oder unter den Tischen.
13. Verlassen Sie Ihren Arbeitsplatz aufgeräumt! Müll gehört in die mehrfach vorhandenen Mülleimer, Altpapier in die dafür vorgesehene blaue Altpapierwanne.
14. Die Fluchtwege sind frei zu halten.

Bei Verstößen gegen die Laborordnung kann die Benutzungsberechtigung versagt werden.

Einleitung

Dieses Dokument soll Ihnen eine kurze Starthilfe in das Mikroprozessorpraktikum geben. Es richtet sich an jene, die noch keinerlei Erfahrung mit hardwarenaher Programmierung haben.

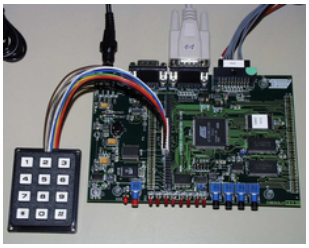
Zuerst werden wir kurz die Zielplattform betrachten, mit der Sie arbeiten werden, sowie die dazu notwendigen Entwicklungswerkzeuge in ihren wichtigsten Grundfunktionen vorstellen.

Bedenken Sie bitte, dass dieses Dokument keinesfalls ausreichend ist, um die Übungsbeispiele zu lösen. Datenblätter sind nun einmal primäre Quellen im Bereich der hardwarenahen Programmierung. Im Mikroprozessorpraktikum kann und soll Ihnen die Beschäftigung damit nicht erspart bleiben. Wir wissen aus eigener Erfahrung, dass es oft mühsam ist, aus den mitunter nicht optimal aufbereiteten einschlägigen Datenblättern die wichtigsten Informationen herauszuholen. Dieses heraus filtern der wesentlichen Informationen in möglichst kurzer Zeit ist aber definitiv eine Fertigkeit, die beim Einstieg ins Berufsleben von Ihnen verlangt werden wird. Sie werden in Ihrer späteren Arbeitsumgebung auch nicht allmonatlich auf Firmenkosten nach Singapur verschifft, um eine persönliche Einführung zum Bauteil des Monats zu bekommen; auch werden Ihre Kollegen mit Ihnen keine Freude haben, wenn Sie die Hardware ständig im Detail erklärt haben wollen.

Andererseits handelt es sich hier um eine Lehrveranstaltung, daher sind Sie natürlich nicht auf sich alleine gestellt. Sie sollen nicht tagelang an ein und demselben Problem sitzen und letztlich frustriert unsere Geräte zum Fenster hinaus werfen. Als erste Anlaufstelle gibt es in den betreuten Übungszeiten Personal, welches Ihnen entsprechende Hinweise geben kann. Bedenken Sie aber bitte, dass es sich bei diesem Personal auch um keine Übermenschen handelt.

Hardware

Um die Programmieraufgaben möglichst vielfältig und abwechslungsreich gestalten zu können, haben wir uns für das Evaluationsboard AT91EB63 der Firma ATMEL entschieden. Durch die Möglichkeit Peripherie an das Board anzuschließen, werden Sie in verschiedenen Versuchen, die Funktionalität der einzelnen Komponenten eines modernen Mikrocontroller besser kennenlernen.



Controllerboard

Das Herzstück der Übungshardware ist das Evaluationboard AT91EB63. Die relevanten Komponenten darauf sind,

- der Kontroller AT91M63200
- zwei serielle Schnittstellen

  Die HOST-Schnittstelle wird zur Verbindung mit einem Rechner zwecks Kommunikationsschnittstelle genutzt.
- ein Resettaster
- vier 4 Taster
- acht Leuchtdioden
- 256KByte statischer Speicher (RAM)
- 2MByte Flash
- 2MByte serielles DatenFlash
- 64KBytes serielles EEPROM
- 2*32-pin EBI Erweiterungsverbinder
- 2*32-pin MPI Erweiterungsverbinder
- 2*32-pin Ein/Ausgabe Erweiterungsverbinder

  Hieran können angeschlossen werden:
  - Über ein Interface eine Saitenschwingwaage
  - Über ein Interface eine Kolbenhubpumpe
  - Ein Tastenfeld mit 3*4 Tasten
  - Eine Box (mit AVR-Mikrokontrollersystem) zur Simulation der Saitenschwingwaage und der Pumpe
- 20-pin JTAG Schnittstellenverbinder

Hieran angeschlossen ist im Labor ein BDI2000 Echtzeitemulatorsystem. Über den BDI2000 wird die Verbindung über das Netzwerk (Ethernet 10Mbit) zum Entwicklungsrechner hergestellt.

Für weitere Informationen schauen Sie im Benutzer Handbuch „AT91EB63 Evaluation Board" nach.

Der Mikrokontroller AT91M63200

Der AT91M63200 ist ein Mitglied der Atmel AT91 16/32-Bit Mikrokontrollerfamilie auf Basis des ARM7TDMI Prozessorkern. Der Prozessor hat eine hochperformante 32-Bit RISC-Architektur, unterstützt den hocheffizienten 16-Bit Befehlssatz (THUMB) und hat eine sehr geringe Verlustleistung. Um nicht alle Besonderheiten des Kontroller hier aufzählen zu müssen, verweisen wir an dieser Stelle auf die technische Dokumentation der Firma

ATMEL
AT91
ARM® Thumb®
Microcontrollers
AT91M63200

Rev. 1028A-11/99

Entwicklungsumgebung ARM-Toolchain und Aufgaben

Im Praktikum zur Rechnerarchitektur lernten wir eine ARM-Toolchain kennen. Mit dieser werden wir auch im Mikroprozessor-Praktikum arbeiten. Zur Wiederholung und Auffrischung beschäftigen Sie sich mit den Aufgaben von Termin 1 zum Praktikum Mikroprozessorsysteme WS2019. Diese Aufgaben können im ARM-Simulator des GDB (arm-eb63-elf-insight) getestet werden. Die Labore D10/0.32 und D10/0.35 stehen Ihnen während der freien Übungszeiten hierfür zur Verfügung.

Sie möchten sich Ihre eigene Entwicklungsumgebung installieren? Dann schauen Sie z.B. mal hier vorbei: http://www.alphapogo.de/
Oder schauen Sie im Wiki z.B. unter
(https://wiki.h-da.de/fbi/technische-systeme/index.php/Mikroprozessorsysteme_Historie#16._Oktober_2012_neues_Debian-Paket)
des Fachbereich Informatik nach.

Zu jedem Termin für das Praktikum Mikroprozessorsysteme WS2019 gibt es einen Ordner mit Informationen und Beispielen. Sie müssen Ihre Praktikumstermine wahrnehmen. Nur im Labor für Mikroprozessorsysteme D10/0.32 steht Ihnen zur Lösung der Aufgaben Termin 2 bis Termin 6 die notwendige Hardware zur Verfügung.

Hinweis zu den Aufgaben / Termine

Die folgend aufgeführten Aufgaben stammen aus einem vorherigen Semester.

Die Aufgaben werden immer weiter entwickelt. Es können in den Aufgaben für das aktuelle Semester also Änderungen vorgenommen worden sein.

Bitte besorgen Sie sich die aktuellen Aufgaben für das Labor bei Ihrem zuständigen Professor oder dem Laboringenieur.

Beachten Sie die Hinweise in den Vorlesungen oder der ersten Laborveranstaltung.

Termin 1

Lernziele:

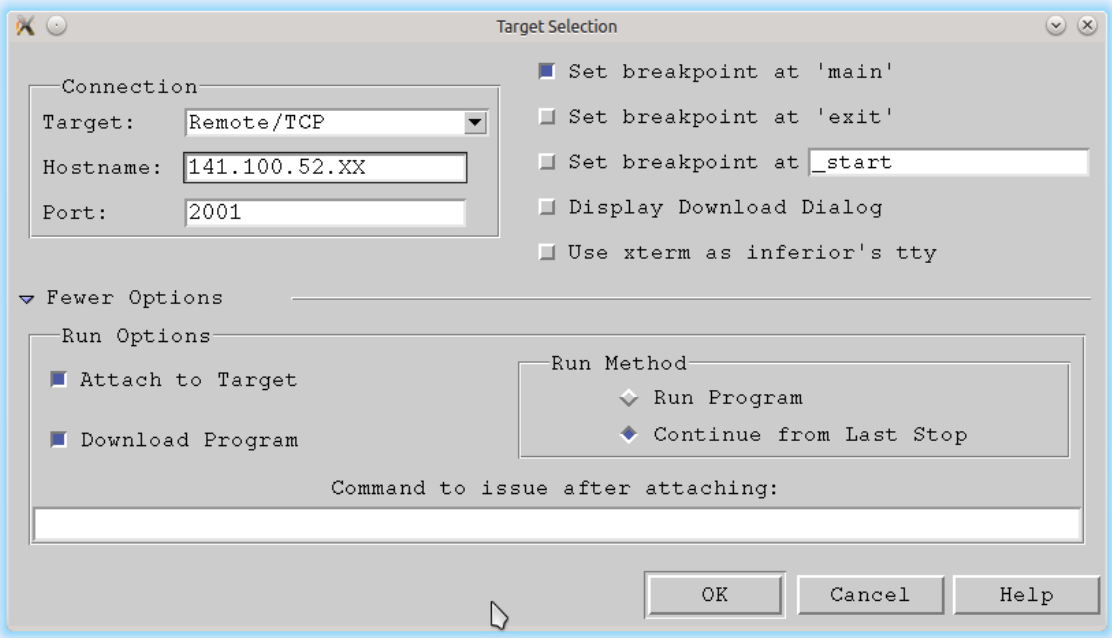
Mit den folgenden Versuchen sollen Sie die Sprache "C" einmal aus einer anderen Sicht kennen lernen. An ganz einfachen Programmen sollen Sie ermitteln, welchen Code ein Compiler erzeugt, wo welche Variablen abgelegt werden, welchen Einfluss die Optimierungsstufen haben, wie ein c*all by value / reference* in ARM Assembler umgesetzt wird. Wie auf Peripherie (System on Chip) zugegriffen wird.

Arbeitsverzeichnis:

Kopieren Sie sich das Verzeichnis /mpsWS2019. Dort stehen im Unterverzeichnis Termin1 die Dateien *Termin1AufgabeX.c als* Programmgerüstbeispiele und *makefile* zur Verfügung.

Infos

Es ist natürlich etwas mühsam/umständlich die nicht vorhandenen LED DS1-DS8, Tastendrücke KEY1 bis KEY3 durch sichten und manipulieren der Registerinhalte zu simulieren. Statt sich mit dem Target Simulator zu verbinden, können Sie im Mikroprozessorlabor auch die ARM7-Evaluationssysteme (AT91EB63) nutzen. Hierzu schalten sie die Steckdosenleiste an Ihrem zum Arbeitsplatz gehörenden System an. Nachdem sich das System mit dem auf Ihrem Arbeitsplatzrechner laufenden TFTP-Server verbunden hat, können Sie sich aus dem Debugger mit dem Target Remote/TCP über den BDI2000 mit dem System (AT91EB63) verbinden. Die Verbindung und die Initialisierung hat funktioniert, wenn aus dem Lauflicht der Dioden ein statisches Leuchten der rechten 3 LED (DS6 bis DS8) wurde. Wählen Sie für den Hostname die IP-Adresse, welche auf dem zugehörigen BDI2000 steht. Als Port wählen Sie die 2001. Weitere Einstellungen entnehmen Sie der folgenden Abbildung.



Aufgabe 1:

Legen Sie im C Programm zwei lokale integer Variablen an. Weisen Sie diesen im Code Werte zu, z.B. 0x1 und 0x2. Übersetzen Sie Ihr Programm und schauen Sie sich den erzeugten Code an. Führen Sie das Programm im Debugger aus. Dokumentieren Sie den erzeugten Code.

Hinterfragen Sie Ihre Beobachtungen z. B. mit:
Wo liegen die Werte der Variablen im Speicher?
Wie kommen die Werte in den Speicher?
Welche Register werden verwendet?
..

Ändern Sie im Makefile die Optimierungsstufe und beobachten und dokumentieren Sie die Veränderungen.

Aufgabe 2:

Verwenden Sie nun statt der lokalen Variablen globale Variablen.
Was ändert sich?
Warum?
Wo liegen die Werte der Variablen im Speicher?
..

Aufgabe 3:

Legen Sie nun zwei globale und zwei lokale integer Variablen an. Weisen Sie den Variablen Werte zu. Machen Sie nun Zuweisungen der Form „global=lokal" und/oder „lokal=global".
Was stellen Sie fest?
Ändern Sie im Makefile die Optimierung und beobachten und dokumentieren Sie die Auswirkungen.
..

Aufgabe 4:

Vereinbaren Sie einen Funktionsprototypen für eine Funktion "int addition(int, int, int)", die 3 integer Parameter erwartet. Rufen Sie diese Funktion im Anschluss an die Zuweisung nach Aufgabe 2 und 3 mit den beiden lokalen und einer globalen Variablen auf. Dokumentieren Sie Ihre Beobachtungen. Ändern Sie im Makefile die Optimierung und beobachten und dokumentieren Sie die Auswirkungen.

Aufgabe 5:

Wir werden uns nun mit den Registern der PIO des hier eingesetzten Mikrocontroller (siehe auch Doku AT91M6320.pdf) beschäftigen.

Name	Adresse	Bedeutung
PIOB_PER	0xFFFF0000	PIOB Port Enable Register
PIOB_OER	0xFFFF0010	PIOB Output Enable Register
PIOB_SODR	0xFFFF0030	PIOB Set Output Data Register
PIOB_CODR	0xFFFF0034	PIOB Clear Output Data Register

Schreiben Sie zunächst den Wert 0x100 ins PIOB_PER und dann ins PIOB_OER. Danach schreiben Sie nacheinander den Wert 0x100 einige Male alternierend ins PIOB_SODR und PIOB_CODR. Dadurch sollte die LED DS1 auf dem Board AT91EB63 an und aus gehen. Testen Sie die Funktion im Debugger mit Einzelschritten aus. Im gegebenen Programmbeispiel Termin1Aufgabe5.c sind verschiedene Möglichkeiten des Beschreibens der Register der PIO gezeigt. Beschäftigen Sie sich mit den verschiedenen Programmiermöglichkeiten.

Welche Variante würden Sie bevorzugen und warum?

Aufgabe 6:

Versuchen Sie nun Ihr Programm auf minimale Programmgröße zu bringen.
Wieviele Byte Speicher benötigen Sie für die Initialisierung und um die LED in einer Endlosschleife blinken zu lassen? Aus wie vielen Assemblerbefehlen besteht Ihr minimiertes Programm?

Aufgabe 7:

Erstellen Sie zu diesem Termin ein Protokoll mit den Lösungen zu den Aufgaben und Ihren Erkenntnissen. Das Protokoll sollen Sie zu den nächsten Terminen vorlegen können. Denken Sie auch daran, dass Sie Programmteile in den nächsten Praktika nochmals benötigen. Beschäftigen Sie sich baldigst auch mit den Aufgaben der nächsten Termine.

Beispiel eines makefile zu Termin1 Aufgabe1

```
# zu verwendete Quelldatei
FILE = Termin1Aufgabe1
# Toolchain
TOOLCHAIN = arm-elf-
# Compiler
COMPILER = gcc
# Linker/Binder
LINKER = ld
# Debugger
DEBUGGER = insight
# Optimierungsstufen
OPTI = 0

# Bauen
all:
# uebersetzen der Quelldatei (FILE)
# Der Schalter -c erzeugt nur die Objektdatei aus der Quelldatei ohne zu binden
# Der Schalter -g in gcc fügt Debugging-Code in die kompilierten Objektdateien ein
# Der Schalter -O gibt die zu verwendete Optimierungsstufe (0..3,s) an
# Der Schalter -I weist gcc an, das Verzeichnis include in den Include-Pfad einzufügen
# Der Schalter -L weist gcc an, das Verzeichnis lib in den Library-Pfad einzutragen.

	$(TOOLCHAIN)$(COMPILER) -c -g -O$(OPTI) $(FILE).c -I ../h

# Der Schalter -S erzeugt eine Assemblerdatei aus der Quelldatei
	$(TOOLCHAIN)$(COMPILER) -S -O$(OPTI) $(FILE).c

# Erzeugen weitere benoetitgen Objektdateien
	$(TOOLCHAIN)$(COMPILER) -c -g -O$(OPTI) ../boot/swi.S -o swi.o -I ../h
	$(TOOLCHAIN)$(COMPILER) -c -g -O$(OPTI) ../boot/boot_ice.S -o boot_ice.o -I ../h

# Binden fuer die RAM-Version
	$(TOOLCHAIN)$(LINKER) -Ttext 0x02000000 boot_ice.o swi.o $(FILE).o -o $(FILE).elf

# Debugger starten
debug:
	$(TOOLCHAIN)$(DEBUGGER) $(FILE).elf

# Aufraeumen
clean:
	rm *.o
	rm *.elf
```

Termin 2

Lernziele:

Der Umgang mit digitalen Ein- und Ausgängen und Unterbrechungen (Interrupt).

Arbeitsverzeichnis:

Kopieren Sie sich aus dem Ordner /mnt/Originale das Verzeichnis mpsWS2019. Dort finden Sie zu jedem Termin vorgegebene Dateien.

Aufgabe 1:

Wir werden uns nach Termin 1 nun nochmals mit den Registern der PIO des hier eingesetzten Mikrocontroller (AT91M63200) beschäftigen. Durch die strukturierte Anordnung der vielen verschiedenen Register der Peripherie des Mikrocontroller ist es möglich durch angelegte Strukturen sich die Programmierung und den Umgang mit der Peripherie zu vereinfachen. Schauen Sie sich Termin2Aufgabe1.c an. Testen und beschreiben Sie die Funktion des Programms. Hierzu schauen Sie sich die Dokumentation zum Board AT91EB63 (AT91EB63.pdf) und die Doku des eingesetzten Mikrocontrollers AT91M63200 (AT91M63200 (Complete).pdf).

Mit welchen Anweisung werden die LED's ein- bzw. ausgeschaltet und warum?

..

Aufgabe 2:

Gut, wir können nun die Leuchtdioden (DS1..DS8) kontrollieren. Ändern und erweitern Sie das Programm so, dass die LED DS1 durch drücken der Taste SW1 eingeschaltet und durch drücken der Taste SW2 ausgeschaltet wird. Sollten die Tasten nicht funktionieren, so überprüfen Sie ob im Power Management Controller (PMC) der Clock für PIOB eingeschaltet ist.

In welchem Register muss welches Bit gesetzt sein/werden, damit der Zustand der Tasten über das Pin Data Status Register (PDSR) erfasst werden können?

Aufgabe 3:

Lassen Sie im nächsten Programm zusätzlich die LED DS2 mit ca. 0,5Hz (Zeitschleife programmieren) blinken.

Wie reagieren Ihre Tastendrücke an SW1 und SW2?

Untersuchen Sie die Funktion ihres Programms auch mit verschiedenen Optimierungsstufen.

..

Aufgabe 4:

Schreiben Sie für die Tasten SW1 und SW2 eine passende Interruptserviceroutine. Erklären Sie die Wechsel der ARM-Betriebsmodi.

Wie reagieren nun Ihre Tastendrücke an SW1 und SW2?

Hat das Bedienen der Tasten Einfluss auf die Blinkfrequenz von DS2?

..

Aufgabe 5:

Erstellen Sie zu diesem Termin ein Protokoll mit den Lösungen zu den Aufgaben und Ihren Erkenntnissen. Das Protokoll sollen Sie zum nächsten Termin vorlegen können. Denken Sie daran, dass einige Programmteile nochmals benötigt werden könnten.

Programmgerüstbeispiele:

```
// Loesung zu Termin 2
// Aufgabe 1
// Namen: _____; _____
// Matr.: _____; _____
// vom :  _____
//

#include "../h/pmc.h"
#include "../h/pio.h"

int main(void)
{
 StructPMC* pmcbase = PMC_BASE;          // Basisadresse des PMC
 StructPIO* piobaseB = PIOB_BASE;        // Basisadresse PIO B
 piobaseB->PIO_PER = ALL_LEDS;
 piobaseB->PIO_OER = ALL_LEDS;

 while(1)
 {
  piobaseB->PIO_CODR = ALL_LEDS;
  piobaseB->PIO_SODR = ALL_LEDS;
 }
        return 0;
}

// Loesung zu Termin2
// Aufgabe 4
// Namen: _____; _____
// Matr.: _____; _____
// vom :  _____
//

#include "../h/pmc.h"
#include "../h/pio.h"
#include "../h/aic.h"..

void taste_irq_handler (void) __attribute__ ((interrupt));

void taste_irq_handler (void)
{

}

int main(void)
{

        return 0;
}
```

Termin 3

Lernziele:

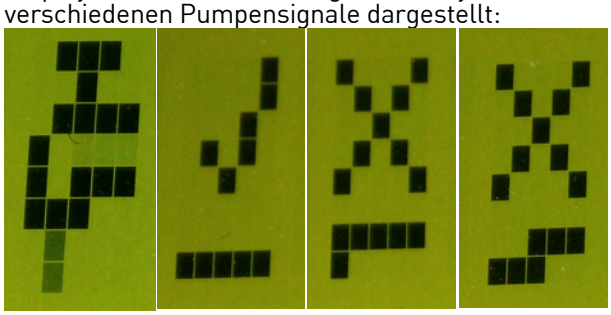
Mit den folgenden Versuchen sollen Sie lernen, wie Sie aus der Sprache "C" Peripherie (z. B. Timer) von modernen Mikrocontrollern nutzen.

Arbeitsverzeichnis:

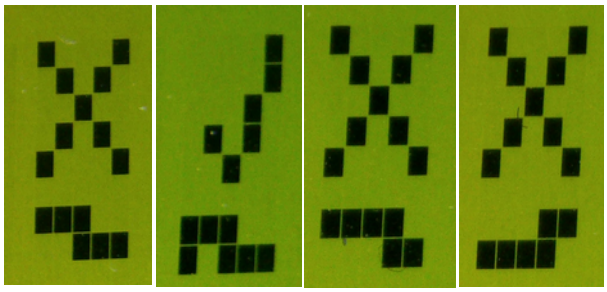
Kopieren Sie sich aus dem Ordner /mnt/Originale das Verzeichnis mpsWS2019. Dort finden Sie zu jedem Termin vorgegebene Dateien.

Weitere Infos:

Ab dem Sommersemester 2011 stehen an jedem Laborarbeitsplatz WaSim (Waagensimulatoren) zur Verfügung. Mit diesen angeschlossenen WaSim kann auch das Pumpensignal überprüft werden. Im Display werden mit folgenden Symbolen die verschiedenen Pumpensignale dargestellt:



es pumpt Gewicht nimmt zu | *kein Signal* | *dauer high* | *Frequenz zu hoch*



Frequenz zu niedrig | *Frequenz richtig* | *Highpegel zu lang* | *Lowpegel zu lang*

Aufgabe 1:

Es soll eine Kolbenhubpumpe, welche über PA1/TIOA3 angesteuert wird, betrieben werden. Die Pumpe benötigt ein symmetrisches Rechtecksignal mit einer Frequenz von ca. 50Hz. Sie könnten eine Zeitschleife programmieren. Hiermit würden Sie aber den Prozessor blockieren (Erinnerung an Termin 2). Besser ist es, Sie initialisieren einen Timer (Timer3) so, dass dieser selbstständig das Signal für die Pumpe erzeugt.

ACHTUNG: Die Pumpe darf kein Dauerhighsignal erhalten.

Vervollständigen Sie die das gegebene Programm *Termin3Aufgabe1.c* entsprechend. Ergänzen und berichtigen Sie auch die Kommentare.

Könnte der eingesetzte Timer bei dem benötigten symmetrischen Signal auch anders betrieben werden?

Zeigen Sie die Berechnung der benötigten Werte um die Frequenz und das Pulsweitenverhältnis richtig einzustellen.

..

Aufgabe 2:

Erweitern Sie Ihr Programm so, dass die Pumpe durch Betätigung von Tasten eingeschaltet und abgeschaltet werden kann.

Welche Möglichkeiten haben Sie gefunden, um das Pumpensignal ein- bzw. auszuschalten?

Für welche Lösung entscheiden Sie sich und warum?

..

Aufgabe 3:

Erstellen Sie zu diesem Termin ein Protokoll mit den Lösungen zu den Aufgaben und Ihren Erkenntnissen. Das Protokoll sollen Sie zum nächsten Termin vorlegen können. Isolieren Sie die Routinen/ Funktionen, welche für die nächsten Termine (siehe speziell Termin6), noch benötigt werden.

Für Fleißige:

Entwickeln Sie eine Funktion, mit der Sie die Tasten, der an einigen Boards angeschlossenen Tastaturen, abfragen können. Mit einer Taste soll die Pumpe eingeschaltet und mit einer anderen Taste soll die Pumpe abgeschaltet werden können.

ACHTUNG: Die Pumpe darf kein Dauerhighsignal erhalten.

Der Anschluss der Tastatur an das Board ist in der Datei **TastaturAnAT91EB63mitBilder.pdf** dokumentiert.

ACHTUNG: Damit die Tastatur verwendet werden kann, ist es nötig den Jumper E4 auf der Entwicklungsplatine (AT91EB63) auf 2 und 3 zu stecken. Siehe hierzu auch im "AT91EB63 Evaluation Board User Guide" auf Seite 6-5.
Infos gibt's auch in der Datei „TastaturAnAT91EB63mitBilder.pdf".
Bitte am Ende Ihrer Sitzung Jumper E4 auf der Entwicklungsplatine (AT91EB63) wieder auf 1 und 2 stecken.

Die Tasten der Tastatur sind nicht entprellt.

```
// Loesung zu Termin 3
// Aufgabe 1
// von:
// vom:
//
#include "../h/pmc.h"
#include "../h/tc.h"
#include "../h/pio.h"
#include "../h/aic.h"

void taste_irq_handler (void) __attribute__ ((interrupt));

// Interruptserviceroutine für die Tasten SW1 und SW2
void taste_irq_handler (void)
{
 StructPIO* piobaseB   = PIOB_BASE;                 // Basisadresse PIO B
 StructAIC* aicbase  = AIC_BASE;                    //__

// ab hier entsprechend der Aufgabestellung ergänzen
// ************************************************


        aicbase->AIC_EOICR = piobaseB->PIO_ISR;         //__
}

// Timer3 initialisieren
void Timer3_init( void )
{
        StructPMC* pmcbase = PMC_BASE;                  // Basisadresse des PMC
        StructTC* timerbase3  = TCB3_BASE;              // Basisadressse TC Block 1
        StructPIO* piobaseA   = PIOA_BASE;              // Basisadresse PIO B

        pmcbase→PMC_PCER = 0x2200;                      // Peripheral Clocks einschalten für PIOA und TC3

        timerbase3->TC_CCR = TC_CLKDIS;                 // Disable Clock

 // Initialize the mode of the timer 3
        timerbase3->TC_CMR =
                TC_ACPC_CLEAR_OUTPUT  |         // ACPC         : Register C clear TIOA
                TC_ACPA_SET_OUTPUT    |         // ACPA         : Register A set TIOA
                TC_WAVE               |         // WAVE         : Waveform mode
                TC_CPCTRG             |         // CPCTRG       : Register C compare trigger enable
                TC_CLKS_MCK1024;                // TCCLKS       : MCKI / 1024

 // Initialize the counter:
        timerbase3->TC_RA = 400;     // hier sind noch die richtigen Werte zu ermitteln
        timerbase3->TC_RC = 800;     // Die Pumpe soll mit einem 50Hz Signal betrieben werden

 // Starten des Timer und Ausgabe eines Low-Pegel an die Pumpe
        timerbase3->TC_CCR = TC_CLKEN ;      //__
        timerbase3->TC_CCR = TC_SWTRG ;      //__
        piobaseA->PIO_PER  = (1<<PIOTIOA3) ;  //__
        piobaseA->PIO_OER  = (1<<PIOTIOA3) ;  //__
        piobaseA->PIO_CODR = (1<<PIOTIOA3) ;  //__
}

int main(void)
{
        StructPMC* pmcbase = PMC_BASE;      // Basisadresse des PMC
        StructPIO* piobaseA = PIOA_BASE;    // Basisadresse PIO A
        StructPIO* piobaseB = PIOB_BASE;    // Basisadresse PIO B


// ab hier entsprechend der Aufgabestellung ergänzen
//************************************************



        return 0;
}
```

Termin 4

Lernziele:

Sie sollen heute lernen wie mit Zählern im Capture-Mode Zeitmessungen realisiert werden können.

Arbeitsverzeichnis:

Kopieren Sie sich aus dem Ordner /mnt/Originale das Verzeichnis mpsWS2019. Dort finden Sie zu jedem Termin vorgegebene Dateien. Ein Schaltplan der Anbindung von Waage und Pumpe finden sie in der Datei **interface.pdf.**

Weitere Infos:

Ab dem Sommersemester 2011 stehen an jedem Laborarbeitsplatz WaSim (Waagensimulatoren) zur Verfügung. Die WaSim liefern zwei, entsprechend des zu simulierenden Gewichtes, sich verändernde Frequenzen im Bereich von ca. 500Hz. Diese beiden Signale sind direkt auf die Timereingänge (über PA7 und PA4) der Evaluierungsplatine AT91EB63 gegeben.

Aufgabe 1:

Machen Sie sich mit dem Programm aus ***Termin4Aufgabe1.c*** vertraut. Übersetzen und testen Sie das Programm. Wie groß ist die gemessene Periodendauer T_{PA4}? Verändert sich die Periodendauer bei Belastung der Waage?

ACHTUNG: Die Waage darf nur mit max. 1000g (1kg) belastet werden!

Aufgabe 2:

Erweitern Sie das Programm so, damit auch die Periodendauer T_{PA7} erfasst wird. Testen Sie auch, ob die Periodendauer sich im zu erwartenden Bereich bewegt.

Aufgabe 3:

Aus dem Verhältnis der beiden Frequenzen f_{PA7}, f_{PA4} kann die Masse **m** nach der Gleichung

$$m = C1 * ((f_{PA7} / f_{PA4}) - 1) - C2$$

oder

$$m = C1 * ((T_{PA4} / T_{PA7}) - 1) - C2$$

errechnet werden. Die Schwingfrequenzen f_{PA7}, f_{PA4} der Saiten liegen bei etwa 16kHz / 32 = 500Hz. Die beiden Größen C1 und C2 sind wägezellenspezifische Konstanten.

Dokumentieren Sie, welche Frequenz mit zunehmendem Gewicht kleiner und welche Frequenz mit zunehmendem Gewicht größer wird.

ACHTUNG: Die Waage darf nur mit max. 1000g belastet werden!

Berechnen Sie die Masse **m** welche auf der Waage liegt. Setzen Sie für C1 und für C2 die für die Waage angegebenen Werte ein. Überprüfen Sie anhand der gegebenen Gewichte ob Ihre Messwerte stimmen.

Aufgabe 4:

Schreiben Sie für die Messung der Masse eine Funktion

int MessungderMasse(void)

welche die ermittelte Masse als integer Wert in g (Gramm) liefert.

Denken Sie auch diesmal daran, dass Programmteile zur Lösung des Gesamtprojektes (Termin6) am Ende wieder benötigt werden.

Aufgabe 5:

Erstellen Sie zu diesem Termin ein Protokoll mit den Lösungen zu den Aufgaben und Ihren Erkenntnissen. Das Protokoll sollen Sie zum nächsten Termin vorlegen können. Teile Ihrer Dokumentation können sicher Bestandteil der Dokumentation des am Ende abzugebenden Projektes sein.

Einige Fragen deren Beantwortung zum Verständnis des Gesamtsystems beitragen:

Wie lange kann eine Messung der Masse dauern?

In welcher Reihenfolge sollten die Messungen durchgeführt werden, um möglichst effizient zu sein?

Können Sie die Aufgabe auch ohne den Einsatz der Bibliothek „libgcc.a" lösen?

...

```
// Lösung zur Aufgabe Termin 4
// Aufgabe 1
//****************************
// Messen der Periodendauer einer angelegten Frequenz
// von: Manfred Pester
// vom: 06. August 2003
// Achtung: Programm ist noch nicht ausgereift!

#include "../h/pio.h"
#include "../h/tc.h"
#include "../h/pmc.h"

// für die Initialisierung des Zähler TC4

#define TC4_INIT  TC_CLKS_MCK2 | TC_LDBSTOP | TC_CAPT | TC_LDRA_RISING_EDGE | TC_LDRB_RISING_EDGE

int main(void)
{
        volatile int captureRA1;
        volatile int captureRB1;
        volatile int capturediff1;
        volatile float Periodendauer1;

        StructPMC* pmcbase = PMC_BASE;
        StructPIO* piobaseA = PIOA_BASE;
        StructTC* tcbase4 = TCB4_BASE;
        StructTC* tcbase5 = TCB5_BASE;

        pmcbase->PMC_PCER = 0x06f80;          // Clock PIOA, PIOB, Timer5, Timer4, Timer1 einschalten

// Periodendauer der Waagensignale messen
// Signal an TIOA4 ca. 16kHz entspricht ca. einer Periodendauer von 62,5us
// durch den Teiler von 32 ergeben sich ca. 2ms
// Zähler mit positiver Flanke starten

        piobaseA->PIO_PDR = 0x090;
        tcbase4->TC_CCR = TC_CLKDIS;
        tcbase4->TC_CMR = TC4_INIT;
        tcbase4->TC_CCR = TC_CLKEN;
        tcbase4->TC_CCR = TC_SWTRG;

        while(!(piobaseB->PIO_PDSR & KEY3))   // ?
        {
                tcbase4->TC_SR;                         // Stati durch lesen des Statusregister zuruecksetzen
                tcbase4->TC_CCR = TC_SWTRG;             // Timer fuer aktuelle Messung starten
                while (!( tcbase4->TC_SR & 0x40));      // Capture Register B wurde geladen Messung abgeschlossen
                captureRA1 = tcbase4->TC_RA;            //
                captureRB1 = tcbase4->TC_RB;
                capturediff1 = captureRB1 - captureRA1;
                Periodendauer1 = capturediff1 / 12.5;   // Zeit in us
        }

        return 0;
}
```

Termin 5

Arbeitsverzeichnins:

Kopieren Sie sich aus dem Ordner /mnt/Originale das Verzeichnis mpsWS2019. Dort finden Sie zu jedem Termin vorgegebene Dateien.

Lernziele:

Die Programmierung von Funktionen die wiederum andere Funktionen aufrufen. Die Kenntnis der Basistechnologie zur Implementierung einer Schnittstelle zwischen Anwendungsprogrammen und Betriebssystemen. Die Bedeutung und Anwendung von Supervisor- und User-Mode.

Aufgabenstellung:

Der SWI Befehl führt zu einer Ausnahmebehandlung im Prozessor die mit einem Wechsel in den Supervisor Mode verbunden ist. Dem SWI Befehl kann beim Aufruf eine bis zu 24 Bit große Zahl übergeben werden, die der SWI Handler dazu benutzen kann die gewünschte Funktion auszuwählen.

In der Aufgabe soll ein SWI Handler genutzt werden, um eine Trennung des Low Level IOs vom Anwendungsprogramm zu erreichen. Dazu sind die Funktionen die den SW-Interrupt aufrufen in Assembler zu implementieren und mit dem Debugger ist die Funktionsweise des Interrupthandlers zu untersuchen.

Aufgabe 1:

Nehmen Sie die zur Verfügung gestellten Dateien in ein neues Projekt auf, testen und dokumentieren Sie dieses.

Die erzeugten Ausgaben von CR und LF sollten auf einem Terminal an der seriellen Schnittstelle zu sehen sein. Verwenden Sie in einer separaten Shell das Programm ***„minicom“*** als Terminalersatz.

Beschreiben Sie den Unterschied der Funktionen inits() und init_ser().

Was passiert, wenn Sie nach CR und LF noch weitere Zeichen auf die gegebene Weise ausgeben?

Aufgabe 2:

Erklären Sie den Code des SWI-Handlers (siehe swi.c/swi.S). Debuggen Sie Ihr Programm und lokalisieren Sie die Umschaltung vom User Mode in den Superuser Mode und zurück.

Woran erkennen Sie, in welchem Mode sich der Prozessor befindet?

An welcher Stelle wird der Superuser Mode verlassen?

Aufgabe 3:

Sie sollen die Funktion puts (siehe ***ser_io.S***) in Assembler so ergänzen, dass auch ein String auf die serielle Schnittstelle ausgegeben wird. Die Initialisierungs-Funktion init_ser und die IO-Funktionen putch und getch (siehe ***seriell.c***) werden dabei über einen SWI (siehe ***swi.c***) aufgerufen.

void puts(char *) Ausgabe eines nullterminierten Strings und Ersetzung von Newline durch Carriage Return (0x0D) und Linefeed (0x0A).

Aufgabe 4:

Entwickeln und **testen** Sie eine Routine, mit der Sie eine vorzeichenbehaftete Integerzahl in einen String wandeln, um diesen dann mit der zuvor entwickelten Routine ***void puts(char *)*** auf ein Terminal ausgeben lassen zu können.

Wie wird die größte darstellbare negative Zahl 0x80000000 ausgegeben?

Aufgabe 5:

Erstellen Sie zu diesem Termin ein Protokoll mit den Lösungen zu den Aufgaben und Ihren Erkenntnissen. Das Protokoll sollen Sie zu den nächsten Terminen vorlegen können. Denken Sie daran, dass zum letzten (sechsten Termin) eine Dokumentation (Funktions- und Programmbeschreibungen, Installationsanleitung, Inbetriebnahme, Benutzerhandbuch) erstellt werden muss.

```
// FileName:  Termin5Aufgabe1.c
// Programmrahmen zur Aufgabe Termin5
// Aufgabe 1
//***********************************
//
// von:
// vom:
// letzte
// von:

int main(void)
{
        char i;
// Serielle initialisieren
        inits();
        init_ser();
// CR und LF auf das Terminal ausgeben
        putc (0xd);
        putc (0xa);
// ein Zeichen von der seriellen abholen
        i=getc();
        while(!(putc(i)));
// String ausgeben
        puts("Hallo! \n");

        return 0;
}
```

```
/*-------------------------------------------------------------------------------------
@ File Name:          seriell.c
@ Object:  Grundfunktionen der seriellen Schnittstelle
@                     int init_ser(); char putch(char); char getch();
@
@ Autor:              M.Pester
@ Datum: 04.12.2007
@-------------------------------------------------------------------------------------*/
#include "../h/pmc.h"
#include "../h/pio.h"
#include "../h/usart.h"

int init_ser(void);
char putch(char);
char getch(void);

#define    DEFAULT_BAUD 38400
#define    CLOCK_SPEED 25000000
//US_BAUD  (CLOCK_SPEED / (16*(DEFAULT_BAUD))          // 25MHz / ( 16 * 38400) = 40.69  -> 41 -> 0x29
#define    US_BAUD 0x29

// Initialisiert die serielle Schnittstelle USART0
int init_ser()
{
        StructPIO* piobaseA = PIOA_BASE;
        StructPMC* pmcbase  = PMC_BASE;
        StructUSART* usartbase0 = USART0;

        pmcbase->PMC_PCER = 0x4;  // Clock für US0 einschalten
        piobaseA->PIO_PDR = 0x18000;            // US0 TxD und RxD
        usartbase0->US_CR = 0xa0;    // TxD und RxD disable
        usartbase0->US_BRGR = US_BAUD;      // Baud Rate Generator Register
        usartbase0->US_MR = 0x8c0;  // Keine Parität, 8 Bit, MCKI
        usartbase0->US_CR = 0x50;    // TxD und RxD enable

        return 0;
}

// Gibt wenn möglich ein Zeichen auf die serielle Schnittstelle aus
// und liefert das Zeichen wieder zurück
// wenn eine Ausgabe nicht möglich war wird eine 0 zurück geliefert
char putch(char Zeichen)
{
        StructUSART* usartbase0 = USART0;

        if( usartbase0->US_CSR & US_TXRDY )
        {
                usartbase0->US_THR = Zeichen;
                return Zeichen;
        }
        else
        {

                return 0;
        }
}

// Gibt entweder ein empfangenes Zeichen oder eine 0 zurück
char getch(void)
{
        StructUSART* usartbase0 = USART0;
        char Zeichen;

        if( usartbase0->US_CSR & US_RXRDY )
                return usartbase0->US_RHR;
        else
                return 0;

}
```

```
@-------------------------------------------------------------------------------
@ File Name:        ser_io.S
@ Object:           Ein- Ausgabe-Funktionen der seriellen Schnittstelle
@                   welche ueber den Supervisor-Mode gehen
@
@ Namen :                   Matr.-Nr.:
@
@-------------------------------------------------------------------------------
@ Debuginformationen
        .file   "ser_io.S"
@ Funktion
        .text
        .align  2
        .global inits
        .type   inits,function
inits:
        swi     0x100
        bx      lr          @ Rücksprung
@ Funktion
        .text
        .align  2
        .global putc
        .type   inits,function
putc:
        mov     r1, r0      @ Zeichen nach r1
        ldr     r0, =Zeichen        @ Adresse der globalen Variablen holen
        str     r1, [r0]    @ Zeichen in globale Variable
        swi     0x200       @
        ldr     r1, =Zeichen        @ Adresse der globalen Variablen holen
        ldr     r0, [r1]    @ Zeichen aus globalen Variable
        bx      lr
@ Funktion
        .text
        .align  2
        .global getc
        .type   inits,function
getc:
        ldr     r0, =Zeichen        @ Adresse der globalen Variablen holen
        swi     0x300
        ldr     r0, =Zeichen        @ Adresse der globalen Variablen holen
        ldr     r0, [r0]    @ empfangenes Zeichen zurueck geben
        bx      lr
@ Funktion
        .text
        .align  2
        .global puts
        .type   puts,function
puts:
        stmfd sp!,{lr}      @ Retten der Register

// Hier muß Ihr Code eingefügt werden.

        ldmfd sp!,{pc}      @ Rücksprung
@ Funktion
        .text
        .align  2
        .global gets
        .type   gets,function
gets:
        stmfd   sp!,{lr}    @ Retten der Register

// Hier könnte Ihr Code eingefügt werden!

        ldmfd   sp!,{pc}    @ Rücksprung

        .data
Zeichen:  .word 0
.end
```

```
/*-------------------------------------------------------------------------
@ File Name:        swi.c
@ Object:           SoftwareInterruptHandler
@
@ Autor:            Horsch/Pester
@ Datum:            3.12.2007/Januar2011
@-------------------------------------------------------------------------*/
void SWIHandler () __attribute__ ((interrupt ("SWI")));

void SWIHandler()
{
        register int reg_r0 asm ("r0");
        register int *reg_14 asm ("r14");

        switch( *(reg_14 - 1) & 0x00FFFFFF)     // Maskieren der unteren 24 Bits

                                                // und Verzweigen in Abh. der SWI Nummer
        {
                case 0x100:
                        init_ser();
                        break;
                case 0x200:
                        *((char *)reg_r0) = putch(*((char *)reg_r0));
                        break;
                case 0x300:
                        *((char *)reg_r0) = (unsigned int) getch();
                        break;
        }
}

# Vorschlag eines Makefile zu Termin5 SS2011

FILE = Termin5Aufgabe1
Opti = 1

all:

# uebersetzen der Quelldatei
        arm-elf-gcc -c -g -O$(Opti) $(FILE).c -I ../h

# Erzeugen einer Assemblerdatei aus der Quelldatei
        arm-elf-gcc -S -O$(Opti) $(FILE).c -I ../h
        arm-elf-gcc -S -O$(Opti) seriell.c -I ../h
        arm-elf-gcc -S -O$(Opti) swi.c -I ../h

# Erzeugen der benoetitgen Objektdateien
# eigener SoftWareInterrupt-Handler
        arm-elf-gcc -c -g -O$(Opti) swi.c -o swi.o -I ../h
        arm-elf-gcc -c -g -O$(Opti) seriell.c -o seriell.o -I ../h
        arm-elf-gcc -c -g -O$(Opti) ser_io.S -o ser_io.o -I ../h
        arm-elf-gcc -c -g -O$(Opti) ../boot/boot_ice.S -o boot_ice.o -I ../h

# Binden fuer die RAM-Version
#       arm-elf-ld -Ttext 0x02000000 -O$(Opti) boot_ice.o swi.o seriell.o ser_io.o $(FILE).o -o $(FILE).elf
/usr/local/arm-elf/lib/gcc/arm-elf/4.3.1/libgcc.a
        arm-elf-ld -Ttext 0x02000000 -O$(Opti) boot_ice.o swi.o seriell.o ser_io.o $(FILE).o -o $(FILE).elf
/usr/local/arm-elf/lib/gcc/arm-elf/4.3.1/libgcc.a

clean:
        rm *.o
        rm *.s
        rm *.elf
        rm *.rom
```

Termin 6

Arbeitsverzeichnis:

Kopieren Sie sich aus dem Ordner /mnt/Originale das Verzeichnis mpsWS2019. Dort finden Sie zu jedem Termin vorgegebene Dateien. Die benötigten Dateien und Programme sollten Sie aus den Terminen 1-5 mitbringen.

Lernziele:

Sie sollen aus dem Gelernten und Gesammelten der Termine 1-5 eine Ausschankstation realisieren.

Die Ausschankstation soll folgende Funktionen haben:

Aufgabe 1:

Initialisierung der benötigten Peripherie.

Aufgabe 2:

Begrüßung und Informationen über die serielle Schnittstelle. Daten zur seriellen Schnittstelle werden im Labor bekannt gegeben.

Aufgabe 3:

Wiegen, kalibrieren und anzeigen des Gefäßgewichtes nach Aufstellen eines Bechers und Drücken der Taste SW1 oder einer Taste der Tastatur des Terminal.

Aufgabe 4:

Abfüllen einer Menge (Menge in Gramm wird im Labor bekannt gegeben) nach drücken der Taste SW2 oder einer Taste der Tastatur des Terminal.
Achtung sollte das Gefäß zu früh weg genommen werden! → Weiter mit Aufgabe 2.

Aufgabe 5:

Nach dem Abfüllvorgang melden welche Menge abgefüllt wurde und das der Becher weg genommen werden kann.

Aufgabe 6:

Wenn der Becher weg genommen ist soll von vorne (Aufgabe 2 evtl. Aufgabe1) begonnen werden.

Aufgabe 7:

- Dokumentieren Sie Ihre Lösung.
- Liefern Sie ausführliche Funktions- und Programmbeschreibungen (Protokolle Termin1 bis Termin5)
- Liefern Sie eine Installationsanleitung.
- Liefern Sie ein Benutzerhandbuch.
- Verkaufen Sie Ihre Lösung dem zuständigen Laborbetreuer (Halten Sie sich an die Vorgaben).
- Sind Sie auf einige Fragen des zuständigen Betreuers vorbereitet.
- Schauen Sie, dass Sie in der Lage sind auf kleine Änderungswünsche reagieren zu können.

Zusatzaufgabe:

Erweitern Sie die Lösung so, dass auch über die Konsole (minicom) die Ausschankstation bedient werden kann. Also die Tasten vom Board nicht mehr benötigt würden.

Termin 6 (wenn nur 5 Termine ohne Termin3)

Arbeitsverzeichnis:

Kopieren Sie sich aus dem Ordner /mnt/Originale das Verzeichnis mpsWS2019. Dort finden Sie zu jedem Termin vorgegebene Dateien. Die benötigten Dateien und Programme sollten Sie aus den Terminen 1-5 mitbringen.

Lernziele:

Sie sollen aus dem Gelernten und Gesammelten einen Münzenzähler realisieren.

Der Münzenzähler soll folgende Funktion haben:

Aufgabe 1:

Initialisierung der benötigten Peripherie.

Aufgabe 2:

Begrüßung und Informationen über die serielle Schnittstelle.

Aufgabe 3:

Wiegen, tarieren und anzeigen des Tara (leeres Gefäßgewicht) über die serielle Schnittstelle, nach Aufstellen eines Gefäßes für die Münzen und Drücken der Taste SW1.

Aufgabe 4:

Information über die serielle Schnittstelle ausgeben, dass nach dem Drücken der Taste SW2 nun Münzen gewogen werden können.

Aufgabe 5:

Wiegen und anzeigen des Nettogewichtes (Gewicht der Münzen) über die serielle Schnittstelle. Anzeigen der Anzahl der Münzen in Binärer Darstellung auf den 8 Leuchtdioden des Board.

Aufgabe 6:

Nach Betätigung der Taste SW3 sollen Tara, Netto und Brutto (Tara plus Netto) in Gramm und die Anzahl der gewogenen Münzen über serielle Schnittstelle ausgegeben werden. Danach weiter bei Aufgabe 2

Aufgabe 7:

- Dokumentieren Sie Ihre Lösung.
- Liefern Sie ausführliche Funktionsbeschreibungen (siehe Protokolle Termin1 bis Termin5)
- Liefern Sie eine Installationsanleitung.
- Liefern Sie ein Benutzerhandbuch.
- Verkaufen Sie Ihre Lösung dem zuständigen Laborbetreuer (Halten Sie sich möglichst an die Vorgaben).
- Sind Sie auf einige Fragen des zuständigen Betreuers vorbereitet.
- Schauen Sie, dass Sie in der Lage sind auf kleine Änderungswünsche reagieren zu können.

Zusatzaufgabe:

Erweitern Sie die Lösung so, dass auch über die Konsole (minicom) die Ausschankstation bedient werden kann. Also die Tasten vom Board nicht mehr benötigt würden.