Universität Kassel
Fachbereich Elektrotechnik/Informatik
Prof. Dr. B. Sick
Fachgebiet Intelligente Eingebettete Systeme

UNIKASSEL VERSITÄT
Intelligent Embedded Systems

Labor Embedded Systems

Intelligente Eingebettete Systeme

Aufgabenblatt 1

Einführung

Betreuer: Benjamin Herwig <bherwig@uni-kassel.de>
und viele großartige Tutoren

Aufgabe 1: Einleitendes

Nehmen Sie vor einem Roboter Platz. Hören Sie dem Betreuer erstmal konzentriert zu und fassen Sie den Roboter nicht an!

Wir werden jetzt in eine Liste eintragen, wer von Ihnen mit welchem Roboter arbeitet. Außerdem unterschreiben Sie ein Regelwerk, das Voraussetzung ist, damit Sie hier bleiben dürfen.

Nachdem das Eintragen in die Liste erfolgt ist, werden wir gemeinsam *Aufgabenblatt 1b* bearbeiten. Erst, nachdem alle von Ihnen damit fertig sind, werden wir dieses Blatt weiter bearbeiten.

Zusätzliche Regel für heute: Kommandozeilenanweisungen und vom Betreuer vorgeturnte Quelltexte brav abtippen, nicht kopieren – zumindest solange nicht, wie Ihnen das nicht anders mitgeteilt wird!

Aufgabe 2: Die Roboter

Nehmen Sie Ihren Roboter, sobald Ihnen das gesagt wird, vorsichtig gemäß Regelwerk in die Hand und schauen Sie ihn sich an. Der Betreuer wird Ihnen in der Gruppe erklären, worum es sich bei den verbauten Komponenten handelt. Folgen Sie allen Anweisungen des Betreuers ganz exakt. Nachdem der Betreuer das ansagt, nehmen Sie den Roboter (mit beiden Händen) und setzen Sie sich an einen beliebigen Rechner im Labor. Stellen Sie den Roboter vor dem Rechner ab. Holen Sie dann auch Ihre Tasche und stellen Sie sie so ab, dass niemand drüber stolpert.

Melden Sie sich am Rechner mit Ihrem Uni-Account an. Eine Auswahlliste mit virtuellen Maschinen wird sich öffnen. Wählen Sie in der Liste das `IES-Labor` und starten Sie es, bspw. über den grünen Pfeil.

Aufgabe 3: Die Rechner

Die Anmeldung im virtuellen Labor erfolgt automatisch.Es handelt sich um **Debian GNU/Linux**-Systeme. Im Startmenü können Sie einen Webbrowser starten, einen Dateimanager öffnen usw.

Wichtig: Bedenken Sie, dass die Daten, die Sie auf dem Rechner ablegen, jederzeit gelöscht werden können. Sorgen Sie also immer(!) für ein Backup Ihrer Arbeit (bspw. per E-Mail, moodle-System, USB-Stick .... `git`, falls Sie mögen und können – zu `git` gibt es auch einen Erklärtext im moodle; zu `git` gibt es jedoch IES-seitig keine Hilfestellung).

Melden Sie sich, sobald Sie fertig sind, laut und deutlich.

Aufgabe 4: Allererste Schritte

Wir werden nun das erste Programmieren der Roboter vorbereiten.

Laden Sie sich alle im moodle unter `Termin 1` angezeigten Dateien (`.c` und `.h`), laden Sie sich außerdem zumindest das Microchip AVR ATMega 328p Datenblatt und das `Roboter-Arduino-AVR.pdf` betitelte Dokument.

Legen Sie Ihre Dateien am besten in einem Ordner ab, den Sie sinnvoll (ggf. mit Unterordnern) benennen; vermeiden Sie bei der Benennung Leerzeichen und Sonderzeichen.

Diesen Ordner könnten Sie dann z. B. immer komplett backupen.

Melden Sie sich, sobald Sie fertig sind, laut und deutlich.

Aufgabe 5: Es geht los – den Roboter programmieren

Der Betreuer wird das folgende einmal „vorturnen“. Schauen Sie ihm zu und konzentrieren Sie sich auf das, was er sagt und zeigt. Danach tun Sie das Folgende:

Öffnen Sie am Rechner `geany` und ein Terminal (beides per Menü erreichbar).

Öffnen Sie in `geany` ein neues Dokument und kopieren Sie dort den aus dem moodle heruntergeladenen Quelltext mit dem Namen `skeleton.c` rein (oder öffnen Sie die Datei direkt in `geany`/aus `geany` heraus).

Sie sehen dann ein C-Programm, das nicht sonderlich viel tut.

Aber übersetzen Sie es, und zwar folgendermaßen:

- Gehen Sie in das Terminal.
- Navigieren Sie auf der Kommandozeile in das Verzeichnis, in dem die Datei `skeleton.c` liegt.
  - Zum Navigieren nutzen Sie den Befehl `cd` (für *change directory*).
  - Durch eingeben von `cd VERZEICHNISNAME` gelangen Sie in das Verzeichnis namens `VERZEICHNISNAME`, beispielsweise in das, das Sie auf dem Desktop angelegt haben (die Desktop-Oberfläche entspricht dem Inhalt Ihres *Home*-Verzeichnisses, in dem Sie schreiben und löschen dürfen).
  - Durch eingeben von `pwd` (für *print working directory*) finden Sie heraus, in welchem Verzeichnis Sie sich gerade befinden.
  - `..` ist ein Kürzel für das *übergeordnete* Verzeichnis.
  - `.` ist ein Kürzel für das aktuelle Verzeichnis.
  - ~ ist ein Kürzel für Ihr *Home*-Verzeichnis.
  - `/` ist ein Kürzel für das Wurzelverzeichnis (*alle* Verzeichnisbäume steigen aus `/` ab. Z. B. ist `/home/pusemuckel` der Pfad zum Home-Verzeichnis des Benutzers Pusemuckel).
  - Mit dem Befehl `ls` zeigen Sie den Inhalt des aktuellen Verzeichnisses an.
  - Mit dem Befehl `ls -al` zeigen Sie den Inhalt des aktuellen Verzeichnisses an, aber inklusive versteckter Dateien und mit allerlei weiteren Informationen zu den Dateien.
  - Mit dem Befehl `ls -al /PFAD_ZUM_VERZEICHNIS/BLA/BLUBB` zeigen Sie den Inhalt des Verzeichnisses `/PFAD_ZUM_VERZEICHNIS/BLA/BLUBB` an, aber inklusive versteckter Dateien und mit allerlei weiteren Informationen zu den Dateien.
  - Mit dem Programm `less` können Sie sich den Inhalt von (Text-) Dateien anzeigen lassen; „hoch“ und „runter“ im angezeigten Text kommen Sie mit dem Pfeiltasten. Im angezeigten Text suchen können Sie durch Tippen von `/` (also `SHIFT+7`), gefolgt von dem Suchstring. Der Suchstring ist *case-sensitive*. Mit dem Befehl `less /usr/lib/avr/include/avr/iom328p.h` zeigen Sie sich z. B. den Inhalt der zu dem auf den Robotern verbauten Microcontroller (Microchip AVR ATmega328p) Header-Datei an. Diese Datei wird noch wichtig (Nur so viel: Inkludieren Sie diese Datei niemals direkt!). Die Anzeige des Texts beenden Sie durch Eintippen von `q`.
  - Hilfe zur Benutzung von Programmen auf der Kommandozeile finden Sie durch die Eingabe des Befehls `man` gefolgt von dem Programmnamen, zu dem Sie gerne Hilfe hätten, z. B. `man less`. Die Anzeige von `man` beenden Sie durch Eintippen von `q` und auch in der *Manpage* können Sie mit `/` nach Strings suchen.
- Compilieren und linken Sie die Datei `skeleton.c`, und zwar durch die Eingabe des Kommandozeilenbefehls `avr-gcc sekeleton.c -o erstes_programm`
- Das sollte ohne Probleme funktioniert haben. Für die Kommandozeile gilt üblicherweise: Sollten Sie keine Fehlermeldung sehen, dann hat alles geklappt. Bestätigen Sie diese Vermutung durch Eintippen von `ls` und verifizieren Sie, dass die Datei `erstes_programm` wirklich im aktuellen Verzeichnis liegt.

- Sie haben soeben Ihr erstes C-Programm für einen Microchip AVR Microcontroller übersetzt. Herzlichen Glückwunsch!
- Die Datei `erstes_programm` liegt in einem Format vor, das nicht geeignet ist, direkt auf den Microcontroller übertragen zu werden.
- Nutzen Sie das Programm `avr-objcopy`. Durch Eingabe des Befehls `avr-objcopy -O ihex erstes_programm erstes_programm.hex` erzeugen Sie eine Datei, die den vom AVR GCC erzeugten Programmcode im sogenannten *Intel Hex Format* enthält.
- Die gerade erzeugte Datei (`erstes_programm.hex`) muss nun noch auf den Microcontroller übertragen werden. Schließen Sie dazu den Roboter per auf dem Rechner liegenden USB-Kabel an den vor Ihnen stehenden Rechner an.
- Nutzen Sie als nächstes das Programm `avrdude`, um die im vorletzten Schritt erzeugte Datei auf den Microcontroller zu übertragen, um die Datei zu *flashen* (also sie in den Flashspeicher des Microcontrollers zu übertragen); geben Sie den folgenden Befehl ein: `avrdude -p atmega328p -c arduino -P /dev/ttyACM0 -b 115200 -U flash:w:erstes_programm.hex:i`. Der Parameterwert `-p` gibt an, welchen Microcontroller wir verwenden; der Parameterwert `-c` gibt an, welchen *Programmer* wir verwenden; der Parameterwert `-P` ist sehr wichtig und gibt an, unter welchem Gerätedateinamen die serielle Verbindung zum Microcontroller angelegt wurde (schauen Sie mal in das Verzeichnis `/dev/`); `-b` gibt an, mit welcher Baudrate (Symbolgeschwindigkeit – recherchieren!) wir mit dem Programmer „sprechen“; `-U` gibt an, wohin (`flash`) wir welche Datei (`erstes_programm.hex`) in welchem Format (`i`) *schreiben* (`w`) wollen.
- **Zusammenfassung:**
  1. `avr-gcc sekeleton.c -o erstes_programm`
  2. `avr-objcopy -O ihex erstes_programm erstes_programm.hex`
  3. `avrdude -p atmega328p -c arduino -P /dev/ttyACM0 -b 115200 -U flash:w:erstes_programm.hex:i`
- Sie sehen nun . . . nicht besonders viel. Ihr Programm endet sofort und (fast) augenblicklich. Nach dem Einschalten des Programms läuft ein kleines „immer“ fest in den Microcontroller geschriebenes Programm, das *Bootloader* genannt wird. Es sorgt z. B. dafür, dass wir Programme mit `avrdude` relativ Problemlos auf den Microcontroller flashen können. Hätten wir den Bootloader nicht, dann wäre das Programmieren des Microcontrollers etwas aufwändiger. Der Bootloader wurde vom „Arduino“-Team geschrieben, das viel Software und Hardware im Bereich des *Open Source* bereit stellt. Unser Microcontroller befindet sich auf einem *Arduino Uno R3*-Board. Die Schaltpläne dazu kann jede*r sich anschauen und jede*r kann die Hardware auch selbst nachbauen. Den Bootloader kann man übrigens auch löschen (oder ersetzen) (was wir aber nicht tun werden!).
- Die oben gezeigten Programmierschritte können Sie im Terminal immer wieder ausführen, und zwar dadurch, dass Sie im Terminalfenster „Pfeil nach oben“ drücken. Einen „kaputten“ Befehl, also wenn Sie sich vertippt haben, können Sie übrigens durch `STRG+C` abbrechen. Mit `STRG+R` können Sie in der Befehls-Historie suchen.

Hausaufgabe: Recherchieren Sie, was das ELF-Format und was in dem Kontext eine Objekt-Datei ist. Was bedeuten die Bezeichnungen `segment` und `.text`? Worum handelt es sich bei einer HEX-Datei (also bei einer Datei, die auf den Microcontroller geflasht wird)?

Aufgabe 6: Eine Leuchtdiode leuchten lassen, oder: Etwas *sinnvolles* programmieren

Spielen Sie die obigen Schritte mit dem Programm `skeleton_led.c` durch. Versuchen Sie *erstmal* nicht, das Programm zu verstehen. Bereits beim Kompilieren werden Sie eine Vielzahl von Fehlermeldungen sehen, wobei bereits die erste Fehlermeldung uns einen Hinweis darauf gibt, was schief läuft:
`/usr/lib/avr/include/avr/io.h:623:6: warning: #warning "device type not defined"`
In der ersten Zeile des zu kompilierenden Programms steht eine Präprozessoranweisung die dafür sorgt, dass die Header-Datei `avr/io.h` eingebunden werden soll. Denken Sie an die Vorlesung zum Präprozessor des

ersten Modulteils: Durch die Spitzklammern sucht der Präprozessor in den *Standardincludepfaden* nach dem Verzeichnis `avr`, und dort wiederum nach der Headerdatei `io.h`.
Schauen Sie sich diese Headerdatei in der Konsole (also: `less/usr/lib/avr/include/avr/io.h`) an und suchen Sie nach der Fehlermeldung. Sie sehen dort (bzw. darüber) eine Menge unterschiedlicher Präprozessoranweisungen. Diese werten eine (zuvor definierte) Präprozessor-Konstante aus, die, falls sie definiert wurde, entscheidet, welchen der vielen verschiedenen AVR-Microcontroller wir programmieren wollen.

Um unserem Präprozessor mitzuteilen, welche Header-Datei inkludiert werden soll, können wir auf der Kommandozeile eingeben, welche *MCU* (*Micro Controller Unit*) wir verwenden wollen, und zwar per Flag:
`-mmcu=atmega328p`, da wir einen ATmega 328p verwenden.

Der jetzt richtige Kommandozeilenaufruf lautet also:
`avr-gcc -mmcu=atmega328p skeleton_led.c -o erstes_sinnvolles_programm`

Denken Sie daran, die Aufrufe für `avr-objcopy` usw. entsprechend anzupassen!

Wenn Sie fertig sind und alles richtig gemacht haben, sollte die grüne, mit `L` markierte Leuchtdiode vorne auf dem Shield dauerhaft leuchten.

Melden Sie sich laut und deutlich, sobald Sie hier angekommen sind.

Aufgabe 7: Programmverstehen des ersten sinvollen Programms

Diese Aufgabe wird der Betreuer mit Ihnen durchgehen. Hören Sie ihm zu, arbeiten Sie diese Aufgabe dann selbstständig als **Hausaufgabe** nochmals durch.

Öffnen Sie das Datenblatt Atmel[1] AVR ATmega 328p und dort Seite 14. Dort sehen Sie ein „Pinout“ des ATmega328p. Öffnen Sie außerdem das PDF-Dokument mit dem Titel `Pinmapping.pdf`.

Melden Sie sich und warten Sie, sobald Sie an dieser Stelle angelangt sind. Wir machen erst weiter, sobald *alle* sich gemeldet haben.

Gehen Sie das letzte Programm Zeile für Zeile durch.

Sie sollten sich fragen: Was geschieht dort? Was sind `DDRB`, `PORTB`, und `PB5`?

Im Schaubild auf Seite 14 sehen Sie den Microcontroller von oben. Dort sehen Sie alle Beinchen des Microcontrollers mit unterschiedlichen Bezeichnungen. Diese Bezeichnungen entsprechen nicht nur unterschiedlichen Beinchen, sondern auch dazugehörenden *Registern* des Microcontrollers. Die Beinchen-Bezeichnungen, die in Klammern gesetzt sind, entsprechen Spezialfunktionen, die wir zum Teil später (beispielsweise bei der Kommunikation mit U(S)ART, bei späterer Analogspannungsmessung usw.) verwenden werden.

Vorerst werden wir uns mit einfachem digitalen Schreiben und Lesen beschäftigen.

Der Microcontroller verfügt über GPIO-Register (General Purpose Input/Output-Register), die in den Gruppen B, C und D zusammengefasst sind (A gibt es bei größeren ATmega-Versionen, z. B. beim ATmega2560). Die GPIO-Register werden (manchmal) auch *`Port`* genannt.

`PB5` ist das Microcontrollerbeinchen, das zum `Port` mit der Bezeichnung *`B`* gehört.

Die einzelnen Microcontrollerbeinchen werden über unterschiedliche interne Register konfiguriert bzw. genutzt.

Jedes Microcontroller-Beinchen entspricht einem Bit in jedem der Register *`DDRx`* (Data Direction Register `x`), *`PINx`* (Port INput [Register]`x`) und *`PORTx`* (das ist das Datenregister, in das man z. B. Ausgabewerte für die Beinchen schreibt, also Einsen oder Nullen – eine 1 entspricht der Versorgungsspannung des Miccrocontrollers (ungefähr 5 Volt), eine Null entspricht der Spannung gegen GND/Masse, also 0 Volt).

Die Buchstaben `B`, `C` und `D` treten jeweils an die Stelle `x` und werden dadurch als zu den entsprechenden Ports gehörend gekennzeichnet.

`DDRB` ist das Daten-Richtung-Register (Data Direction Register) für den Port B. Wenn man Bit 5 in diesem Register auf 1 setzt, so ist das Beinchen PB5 auf Ausgang geschaltet. Wäre `DDRB` an der Bitstelle 5 auf 0 gesetzt, so wäre PB5 ein Eingang.

[1] Das Unternehmen Atmel wurde 2015 von Microchip aufgekauft. Atmel und Microchip werden hier/in der Literatur/im Netz teils synonym verwendet. Eine sehr berühmte Microcontroller-Reihe von Microchip trägt den Namen *PIC* – das sollten Sie mal gehört haben.

`PORTB` ist das Register, in das man für den gesamten Port B Ausgabebits schreibt. Wenn Bitstelle 5 auf 1 gesetzt ist, liegt – solange DDR passend als Ausgang geschaltet ist – liegt an dem Beinchen PB5 VCC an, also eine logische 1. Fall Bitstelle 5 auf 0 gesetzt ist, liegt an PB5 eine logische 0 an.

`PINB` ist das Register, aus dem man für den gesamten Port B Eingaben lesen kann. Wenn Bitstelle 5 auf 1 gesetzt ist, liegt – solange DDR passend als Eingang geschaltet ist – liegt an dem Beinchen PB5 VCC an, also eine logische 1 – ein Sensor könnte das Beinchen z. B. auf VCC gezogen haben. Fall Bitstelle 5 auf 0 gesetzt ist, liegt an PB5 eine logische 0 bzw. GND an. **Wichtiger Hinweis**: Man kann – was wir aber im Praktikum nicht tun werden, da wir das nicht brauchen – auch in PIN`x` schreiben, was dem Aktivieren eines sog. *Pull-Up-Widerstands* an diesem Beinchen entspricht. Wir beschäftigen uns damit hier nicht weiter, es gibt zu dieser an und für sich wichtigen Thematik jedoch einen kurzen Erklärtext im moodle. Lesen Sie ihn bei Gelegenheit nach dieser Übung.

Intern sind alle Register(inhalte) über *Adressen* zu erreichen. Die Adressen sind in *jedem* Microcontrollermodell der *selben Baureihe* stets gleich. Aber unterschiedliche ATmegas können die verschiedenen Register an unterschiedlichen Adressen ablegen. Unter anderem deswegen ist es so wichtig, dass beim Aufruf von `avr-gcc` der verwendete Mikrocontroller bekannt ist.

Die konkreten Adressen für den ATmega328p findet man – natürlich – im Datenblatt. Schauen Sie dazu auf Seite 428. Dort sehen Sie, dass die Registeradressen für die den verschiedenen Ports zugeordneten Register ab 0x23 aufeinander folgen. (Hätten Sie übrigens einen ATmega2560 fände sich an 0x20 PINA, an 0x21 DDRA und an 0x22 PORTA).[2]

Diese Adressen sind *dereferenzierbar*, d. h. man kommt an den „Speicherinhalt“ hinter den Adressen über den Dereferenzierungsoperator. Das Dereferenzieren geschieht über *Präprozessor-Makros*. Denken Sie an die Vorlesungen zu Pointern und Präprozessormakros.

Sie könnten z. B. versuchen, den Bezeichner `DDRB` als Bezeichner zu interpretieren, und die Anweisung in Zeile 5 als schreiben eines Werts „an die Speicherstelle, die von diesem Bezeichner bezeichnet wird“ interpretieren.

Tatsächlich geschieht da aber erheblich mehr. Bei `DDRB` handelt es sich um ein Makro (bzw. um eine Präprozessorkonstante). Sie ist in der Datei `/usr/lib/avr/include/avr/iom328p.h` definiert, und zwar mit dem Wert `_SFR_IO8(0x04)`. Dieses Makro ist wiederum definiert in `/usr/lib/avr/include /avr/sfr_defs.h` per `#define _SFR_IO8(io_addr) _MMIO_BYTE((io_addr) + __SFR_OFFSET)`.

`__SFR_OFFSET` ist eine Präprozessorkonstante, die den Wert `0x20` hat – sie ist dafür zuständig, dass die Adressanordnung der ATmega-Microcontroller berücksichtigt werden (da von `0x00` bis einschließlich `0x1F` sogenannte *General Purpose Register* liegen, auf denen die *ALU* des Microcontrollers direkt und rasend schnell arbeiten kann – der ATmega hat 32 (also 20_{16}) dieser Register, die uns hier aber nicht interessieren). Man macht das, weil man manchmal (z. B. iVm. Assembler-Programmierung) zwischen RAM-Adressierung und „AVR-Adressierung“ wechseln möchte.

Jetzt nähern wir uns dem Ziel … `_MMIO_BYTE(ZAHLENWERT)` ist in der selben Header-Datei definiert per `#define _MMIO_BYTE(mem_addr) (*(volatile uint8_t *)(mem_addr))`.

Das bedeutet, dass das Makro `DDRB` zu folgendem Ausdruck expandiert: `(*(volatile uint8_t *)(0x24))`.

Der Compiler dereferenziert also die Adresse `0x24` und macht dem Rechner außerdem klar, dass auf eine Speicherstelle, die als `uint8_t` – das ist vergleichbar mit einem `unsigned char`, also einem Byte, wobei es sich um eine Typ(um)definition handelt, die der Klarheit dient – interpretiert wird, zugegriffen werden soll.

An die Speicherstelle hinter der Adresse wird also der Wert geschrieben, der rechts vom Gleichheitszeichen in Zeile 5 steht. Da ist eine (einzige) Bitstelle auf 1 gesetzt. Für jede Bitstelle, die im `DDR` auf 1 gesetzt ist, wird das entsprechende Beinchen als Ausgang geschaltet. Wenn eine Bitstelle im DDR auf 0 gesetzt wird, ist das entsprechende Beinchen als Eingang konfiguriert. Die Konstante `DDB5` findet sich auch in der Header-Datei, in der `DDRB` definiert war. Es handelt sich um einfache Zahlenkonstanten, die als Präprozessorkonstante definiert sind. Sie enthalten die Zahlenwerte, die im Datenblatt des Mikrocontrollers zu finden sind. Dort jedoch als `DDRBn`, in den Headerdateien als `DDBn`. Es ist absolut notwendig, diese Konstanten zu verwenden, da sich die

[2] Im Praktikum (und nur in diesem Praktikum, niemals in der Theorievorlesung) dürfen Sie Hexadezimalwerte auch in Texten mit `0x` notieren. Sie sollten sich aber nicht für Ihren üblichen Lebensalltag(?) angewöhnen, Hexadezimalwerte auf diese Art anzugeben; das ist eine nicht immer definierte Schreibweise (denn spätestens bei der *Endianess* und wenn man mehr als ein Byte notieren möchte, knallt das richtig … aber das ist ein anderes Thema). Schreiben Sie die Basis des Zahlensystems immer an die notierte Zahl, wenn nötig.

Zahlenwerte ja irgendwann mal ändern könnten, z. B. wenn eine neue Mikrocontrollergeneration auf den Markt kommt und der Hersteller die Pinanordnung ändern möchte – das ist besonders dann wichtig, wenn man die Spezialfunktionen der verschiedenen Beinchen nutzen möchte, die z. B. über Aus- oder Eingabe hinaus gehen.

In Zeile 5 wird also das Beinchen 5 am Port B auf „Ausgang“ geschaltet. Vergleichen Sie Die Portnamen (B, C, D) mit den Arduino- bzw. Elegoo-Shield-Bezeichnern – Sie finden diese Information in `Pinmapping.pdf`. In der rechten Spalte sehen Sie, welche roboter-Peripherie-Funktion am jeweiligen Microcontroller-Beinchen angeschlossen ist.

In Zeile 7 wird nun wieder durch die Headerdateien gesprungen wie zuvor und an die Adresse `0x25` (Datenregister für Port B, vgl. Datenblatt) ein Byte geschrieben, in dem ein Bit auf 1 gesetzt ist. Dadurch wird das entsprechende Beinchen des Port B auf/gegen VCC „gezogen“, d. h. dass eine Spannung anliegt und es zu einem Stromfluss über die an dieses Beinchen angeschlossenen LED kommt. Die LED ist über einen Vorwiderstand angeschlossen.

Auch hier endet Ihr Programm nach dem Schalten des Beinchens `PB5` auf die Versorgungsspannung.

Benutzen Sie beim Programmieren immer die jeweils in der soeben besprochenen zum Microcontroller gehörenden Header-Datei angegebenen Konstanten, die unter jeder Register-Konstante zu finden sind, um die einzelnen Register-Stellen anzusprechen. Für `PORTx` also z. B. `PORTx1`, für `DDRx` entsprechend z. B. `DDx1`, und für `PINx` z. B. `PINx1`.

Hausaufgabe:
Recherchieren Sie und beantworten Sie für sich schriftlich in 5 Sätzen, wie eine LED (Leuchtdiode) funktioniert. Nehmen Sie dazu beispielsweise den entsprechenden Erklärtext im moodle, aber auch weitere Ergänzungsliteratur.

Aufgabe 8: Eine Leuchtdiode blinken lassen

Laden Sie das Programm `skeleton_led_blink.c` in `geany` und übersetzen Sie es wie in der eben bearbeiteten Aufgabe im Terminal.

Sie werden zwar keinen Fehler sehen, aber zwei Warnungen.

Im Programm sollen die Leuchtdioden für 500 Millisekunden leuchten, dann für 500 Millisekunden nicht leuchten, dann ... usw.

Um das zu realisieren, wurde eine Endlosschleife in das Programm eingebaut. Das sorgt dafür, dass Ihr Programm niemals endet.

Um für die jeweiligen Zeitspannen zu warten, wird die Funktion `_delay_ms(double)` verwendet. In dieser Funktion wird eine Schleife aufgerufen, die per einer *busy waiting* genannten Methode Rechenzeit des Mikrocontrollers verbrät. Dazu wird eine bestimmte Anzahl von Anweisungen abgearbeitet, die abhängig von der Taktfrequenz des Mikrocontrollers unterschiedlich lange zum Erledigen benötigen. Damit die Wartefunktion also richtig arbeiten kann, wird eine Information darüber benötigt, mit welchem Takt der Mikrocontroller betrieben wird. In unserem Fall sind das 16 Megahertz (was schon ziemlich ziemlich schnell ist – die meisten Maschinenbefehlanweisungen im Mikrocontroller können innerhalb eines Takts abgearbeitet werden).

`_delay_ms(double)` finden wir in der Header-Datei `util/delay.h`. Falls wir nicht *vor* dem Inkludieren dieser Datei irgendwo im Code (oder mit einem Kommandozeilenflag für den Präprozessor) eingestellt haben, wie schnell unser Mikrocontroller arbeitet, kann die Funktion nicht korrekt arbeiten, da sie alle Timing-Informationen aus der Prozessorgeschwindigkeit berechnet.

Sie können im Code also die Zeile 2 *einkommentieren*. Welchen Zahlenwert hat dann die Präprozessorkonstante `F_CPU`, was bedeutet `16E6`? Alternativ können Sie auch Ihren Kommandozeilenaufruf anpassen ... wie? (Tipp: `-D`)

Sie *müssen* darauf achten, dass Sie `F_CPU` keinesfalls nach dem Inkludieren von `util/delay.h` definieren, denn dann besteht die Warnung weiterhin.

Die verbleibende Warnung zeigt an, dass wir noch *Compileroptimierungen* anschalten müssen, damit die Wartefunktion sich auf bestimmte Kompilateigenschaften verlassen kann. Der Compiler erkennt z. B Coderedundanzen oder Abschnitte, die nie betreten werden oder auch Bereiche, die durch andere Anweisungen als angedacht

schneller gelöst werden können. Es gibt unterschiedliche Optimierungsstufen. Allen gemeinsam ist, dass das Kompilat nach dem Optimieren erheblich schwerer zu *debuggen* (jedoch ist hier eine Art von debugging gemeint, mit der wir uns nicht beschäftigen werden) ist. Die für Mikrocontroller übliche Optimierung ist die in Hinblick auf eine möglichst geringe Kompilatgröße (denn der Speicherplatz für das Programm ist irgendwann immer zu knapp). Auf der Kommandozeile können Sie beim Compilieren das Flag `-Os` mit angeben, um die Größenoptimierung anzuwerfen. (Idee: Vergleichen Sie doch mal im Terminal die Größen der erzeugten Kompilate mit und ohne Optimierung).

Nachdem Sie also mit `F_CPU` und `-Os` kompiliert und geflasht haben, sollten Sie eine blinkende Leuchtdiode auf dem Shield sehen. Ändern Sie Ihren Code und variieren Sie die Blinkfrequenz.

Wichtiger Hinweis: Das Argument, dass Sie an `_delay_ms` übergeben, muss unbedingt schon zur Compile-Zeit bekannt sein. Sie dürfen Berechnungen, falls Sie welche durchführen wollen, nur mit dem Präprozessor durchführen. Sie dürfen kein variables Argument, also keinem Wert der sich während der Laufzeit verändern bzw. zur Compile-Zeit nicht feststeht, als Argument übergeben.

Aufgabe 9: Eine erste, semiclevere Automatisierung des Softwarebuilds

Denken Sie daran, dass Sie die Befehle im Terminal durch „Pfeil hoch“ einfach wiederholen können.

Alternativ können Sie sich auch ein Shellscript schreiben:

Legen Sie mit `geany` eine Datei an, die Sie `compile.sh` (oder so, aber die Endung ist „Standard“ und steht für `shellscript`) nennen.

Tippen Sie dort alle Befehle ein, die Sie bisher immer benutzt haben (NICHT KOPIEREN!). In die erste Zeile *müssen* Sie jedoch die „Shell“ bzw. den Interpreter für Ihre Kommandos eintragen. Üblicherweise ist das die `bash`, die *Bourne-Again SHell*. Das machen Sie über die Anweisung (in der *ersten* Zeile) `#!/bin/bash`.

Speichern Sie die Datei dann.

Sie können ein Shellscript z. B. dadurch ausführen, dass Sie im Terminal in dem Verzeichnis, in dem auch das Shellscript liegt `./compile.sh` tippen.

Sie werden dann eine Fehlermeldung erhalten. Man darf nicht „einfach so“ Dateien ausführen, sondern muss das sog. *executable bit* setzen. Das erledigen Sie durch `chmod +x compile.sh`.

Danach können Sie das Script ausführen und alle Schritte (kompilieren, objcopy, flashen) werden nun automatisch durchgeführt.

In einer der folgenden Veranstaltungen werden wir eine andere Art der Automatisierung des Software-Builds kennen lernen.

Falls es Ihnen zu anstrengend ist, selbst ein shell-Script zu schreiben, so können Sie auch die `compile.sh`-Version nehmen, die im moodle im Bereich `Tools` hochgeladen wurde. Bitte beschäftigen Sie sich jedoch mit der Shell-Programmierung. Mindestens ist es Ihnen selbst überlassen, zu verstehen, wie Sie das vorgegebene Shell-Script benutzen und auch herauszufinden, was es mit `$1,$2,...` auf sich hat

Aufgabe 10: Kommunikation per U(S)ART – mit der Welt reden

Nach dem Flashen eines HEX-Files kann der Microcontroller das USB-Kabel nutzen, um darüber mit dem angeschlossenen Computer zu kommunizieren. Effektiv ist über eine auf den Microcontrollerträgerboards verbaute Elektronik der `U(S)ART`-Anschluss mit dem USB-Anschluss verbunden.

Alle Daten, die Sie über das Beinchen, das mit TXD beschriftet ist (vgl. Pinout auf Seite 14 im Datenblatt), ausgeben, werden an den angeschlossenen Computer gesendet. Alles, was Sie vom Rechner an den Microcontroller senden, kommt am mit RXD bezeichneten Beinchen an. TXD und RXD sind Spezialfunktionen zweier Beinchen an einem Port (Port D). Diese Beinchen könnten, wenn sie nicht für U(S)ART verwendet würden, auch als GPIOs genutzt werden. Dann hätten wir aber nicht mehr die Möglichkeit, den Microcontroller einfach über ein USB-Kabel direkt auf dem Arduino-Board zu flashen.

Mit den Logikpegeln an den Beinchen brauchen Sie sich nicht selbst zu arbeiten. Verwenden Sie die Dateien `iesusart.h` und `iesusart.c`. Schauen Sie in die Header-Datei und verwenden Sie die dort deklarierten Funktionen, um folgendes zu erreichen:

1. Initialisieren Sie das USART-Register mit der richtigen Baudrateneinstellung, mit dem richtigen Übertragungsformat. Hat ein sehr netter Mensch Ihnen vielleicht schon viel Arbeit abgenommen? Schauen Sie durch die Header-Datei.
2. Nutzen Sie die Funktion `USART_print(cont *char)` um einen beliebigen String(!) zu übertragen.

Also: Erzeugen Sie eine neue `.c`-Datei und inkludieren Sie den Header iesusart.h – denken Sie aber daran, dass die Header-Datei im selben Verzeichnis wie Ihre neu angelegte Datei liegen muss (oder sie müssen beim Inkludieren in den doppelten Hochkommata mindestens den relativen Pfad zum Header angeben). Dann setzten Sie bitte die in der Liste stehenden Schritte um.

Passen Sie beim Kompilieren auf! Sie müssen *beide* `.c` Dateien übersetzen! Erinnern Sie sich an die Vorlesung: Mit der Compiler-Option `-c` hat man mehrere Objektdateien erzeugt, die man dann (ohne weitere Option) in einem nächsten Schritt vom Linker hat „zusammenlinken“ lassen. Das müssen Sie hier auch tun. Sie müssen, falls Sie ein Shellscript nutzen, also drei Compileraufrufe einbauen. (In Wahrheit kann man auch alles in einem Rutsch erledigen … spielen Sie rum. Versuchen Sie doch den Compileraufruf mit zwei Dateinamen. Denken Sie aber an eine `-o`-Option.)

Um die vom Microcontroller gesendeten Zeichenketten sehen zu können, brauchen Sie ein serielles Terminalprogramm. Starten Sie auf Ihrem Rechner `cutecom`. Sollten Sie das `cutecom`-Icon im Startmenü nicht finden, dann starten Sie `cutecom` keinesfalls aus der Konsole heraus. Sie können z. B. mal `F12` drücken und eine weitere praktische Konsole „rutscht“ auf Ihren Bildschirm. Dort können Sie `cutecom &` eintippen. Das `&` sorgt dafür, dass der gestartete Prozess in den Hintergrund gesendet wird und Ihre Kommandozeile nicht blockiert wird.

In `cutecom` können Sie als „Device“ das wählen, auf das Sie mit `avrdude` geschrieben haben, allermeistens ist das `/dev/ttyACM0` (vielleicht auch mal mit einer anderen Nummer, falls Sie das USB-Kabel schnell abgezogen und wieder angesteckt haben).

Bevor Sie auf `Open` klicken, müssen Sie sicherstellen, dass die `cutecom`-Einstellungen (Button `Settings`) korrekt sind. Gleichen Sie sie mit den Daten, die Sie der Header-Datei entnehmen können, ab.

Sie sollten nun den String, den Sie abgeschickt haben, sehen können.

Falls Sie nichts sehen:

- Kabel eingesteckt?
- Einstellungen im cutecom richtig? Belassen Sie alles bei den Standardeinstellungen.
- Endlosschleife vergessen? Dann waren Sie mit dem Herstellen der Verbindung vielleicht (wahrscheinlich!) zu langsam. Drücken Sie mal auf das Reset-Knöpfchen am Microcontroller-Shield, während die Cutecom-Verbindung noch geöffnet und das Kabel angeschlossen ist.

Falls Sie davon ernsthaft überrollt sind – aber nur dann(!) – können Sie die Datei `skeleton_usart.c` laden. Sie müssen sie aber derart ändern, dass zumindest die Ausgaben per serieller Schnittstelle korrekt sind und genau verständlich ist, welcher Linienfolger gerade feuert usw.

Hausaufgabe:
Recherchieren was U(S)ART ist! Verwenden wir auf den Microcontrollern USART oder UART? Das Pinmapping des Roboters hilft Ihnen bei der Beantwortung dieser Frage! Recherchieren Sie außerdem, worum es sich bei den Begriffen *Baudrate*, *Stop-Bit* **Hausaufgabe:**
Senden Sie vom Rechner ein Zeichen an den Microcontroller und lassen Sie es zurücksenden. Entwickeln Sie quasi ein serielles `Echo`-Programm. Erweitern Sie dieses Programm dann für beliebig lange Strings (wie lang dürfen Ihre Strings aber maximal werden?).

Aufgabe 11: Digitales Lesen

Natürlich kann man auch Werte (in der Regel aus der echten Welt) in den Mikrocontroller einlesen – das geschieht über Sensoren, beispielsweise über die Linienfolgermodule unter dem Roboter.

Die Linienfolgemodule sind drei mal vorhandene, gleiche Kombinationen aus Infrarot-Leuchtdiode und Infrarot-Phototransistor. Recherchieren Sie, wie ein *Transistor* grundlegend funktioniert, beachten Sie bspw. auch den

Erklärtext, den Sie im moodle dazu finden. Vergleichen Sie Ihr Wissen dann mit dem Datenblatt `TCRT5000 Datenblatt.pdf`, das Sie im moodle finden. Kurz zusammengefasst: Ein kleiner Steuerstrom kann einen größeren Strom schalten. Der kleine Steuerstrom wird bei den konkreten im Robotrer verbauten IR-Sensoren über den photoelektrischen Effekt an der Basis des Transisotrs erzeugt. Je mehr Infrarotlicht von der Infrarot-LED zum IR-Phototransistor zurückreflektiert wird, desto geringer ist die Spannung, die über der Collector-Emitter-Strecke des Phototransistors abfällt. Dieser Spannungsabfall ist entweder eher nah an GND oder nah an VCC.

Finden Sie heraus, welcher Linienfolger an welchem Beinchen angeschlossen ist (`Pinmapping.pdf`).Sie können von diesen Eingängen lesen, indem Sie die entsprechenden Beinchen an dem Port als Eingang konfigurieren.

Setzen Sie die entsprechenden Bits im DDR (Data Directyion Register) des Ports *explizit* auf 0 – löschen Sie die Bits an den entsprechenden Stellen also, verlassen Sie sich niemals(!) darauf, dass die Bits auf 0 stehen. Niemals!(!!)

Sie können dann prüfen, ob ein Bit am entsprechenden Beinchen des Ports *x* gesetzt ist, indem Sie das `PINx`-Register überprüfen. Erzeugen Sie sich eine Bitmaske, die Sie mit `PINx` ver-und-en; das Ergebnis der Konjunktion prüfen Sie dann auf „größer 0“ (also so, wie Sie es in der Vorlesung kennen gelernt haben). Falls das gilt, dann ist der Eingang auf logisch 1 gezogen, oder anders ausgedrückt: Der Linienfolge, der an diesem Beinchen angeschlossen ist, „feuert“ bzw. die Spannung ist näher an VCC als an GND.

Falls der Linienfolger feuert, geben Sie die entsprechende Information per serieller Schnittstelle aus.

Zum Prüfen der `PINx`-Registers nutzen Sie bitte unbedingt die Konstanten `PINx[0:7]`.

Nutzen Sie den Quelltext `skeleton_led_blink.c` als Grundgerüst und denken Sie daran, die U(S)ART-Schnittstelle zu initialisieren.

Zum einfachen Testen der Linienfolger können Sie das an Ihrem Arbeitsplatz liegende Trackstück verwenden.

Falls Sie davon ernsthaft überrollt sind – aber nur dann(!) – können Sie die Datei `skeleton_lfin_serout.c` laden. Sie müssen sie aber derart ändern, dass zumindest die Ausgaben per serieller Schnittstelle korrekt sind und genau verständlich ist, welcher Linienfolger gerade feuert usw.

Hausaufgabe:
Wie weiter oebn schon geschrieben. Recherchieren Sie unbedingt, wie ein Transistor (zumindest konzeptuell) funktioniert. Nutzen Sie die Erklärtexte, die Sie im moodle finden, aber auch Ergänzungsliteratur.

Aufgabe 12: Schieberegister und Leuchtdioden

Melden Sie sich laut und deutlich, wenn Sie hier angekommen sind. Der Betreuer wird Ihnen das Grundkonzept dieser Aufgabe vorturnen. Wir machen erst weiter, sobald *alle* an dieser Stelle angekommen sind. Helfen Sie einander daher unbedingt!

Auf dem Roboter ist sehr viel Peripherie-Hardware verbaut. Dabei handelt es sich beispielsweise um die H-Brückenschaltung für den Antriebsstrang, die insgesamt sechs Beinchen des Microcontrollers belegt. Außerdem belegt das Ultraschallmodul zwei Beinchen. Dann gibt es noch die serielle Schnittstelle, eine Reset-Leitung, einen Spannungsmesseingang für die Batterien und einen Servomotor (der belegt nur ein einziges Beinchen).

Einige der Hardwarekomponenten (eine Intertialmesseinheit (IMU) und ein RFID-Lesegerät) sind über ein spezielles Bus-System an den Microcontroller angeschlossen, das Beinchen spart – es handelt sich um I2C. Mit I2C werden wir uns in diesem Praktikum nicht beschäftigen, das ist Inhalt der Folgeveranstaltung *Intelligente Technische Systeme* (ITS).

Außerdem sind auf den Robotern auch noch drei Leuchtdioden verbaut. Um die drei Leuchtdioden zu betreiben – um mit ihnen z. B. die Zustände der Linienfolgermodule zu signalisieren – standen am Microcontroller nach dem Anschließen all der anderen Peripherie nur noch zwei Beinchen zur Verfügung. Um trotzdem drei Leuchtdioden anschließen zu können, aber ohne das recht komplizierte I2C oder eine andere aufwändigere Buslösung wie SPI zu nutzen, wurden die Leuchtdioden über ein Schieberegister an den Roboter angeschlossen.

Die (theoretische) Funktionsweise eines Schieberegisters haben Sie bereits im ersten Semester in der Veranstaltung Digitale Logik kennen gelernt. Öffnen Sie das Datenblatt zum Baustein 74HC4015 im moodle. Dort sehen Sie, dass ein Schieberegister tatsächlich (im wesentlichen) nur aus aneinhander gehängten Schieberegistern besteht. Öffnen Sie außerdem erneut das `Pinmapping`-Dokument. Die dort gelistete Funktion (des Roboters bzw. der Roboter-Hardware) mit der Bezeichnung `SR_CLK` beschreibt den Takteingang des auf dem Roboter verbauten Schieberegisters. Es ist an das Microcontroller-Beinchen `PD4` angeschlossen. Der Dateneingang des Schieberegisters entsprechend an `PB2`.

Machen Sie sich klar, welche der beiden Beinchen `PB2` und `PD4` Sie wann und *warum* beschalten müssen, um eine bestimmte Leuchtdiode leuchten zu lassen.

Schreiben Sie ein kleines Programm, mit dem Sie

1. (nur) die erste (blaue) am Schieberegister angeschlossene Leuchtdiode leuchten lassen,
2. dann nach einer Sekunde (nur) die zweite (grüne) Leuchtdiode und
3. nach einer weiteren Sekunde die dritte (gelbe) Leuchtdiode, um dann
4. nach einer weiteren Sekunde alle Leuchtdioden gleichzeitig leuchten und schließlich
5. nach noch einer weiteren Sekunde alle Leuchtdioden verlöschen lassen.

Erweitern Sie Ihr Programm schließlich derart, dass die Schritte 1 bis 5 in einer Endlosschleife immer wieder durchlaufen werden.

Tipp:
Nutzen Sie die serielle Ausgabe zum einfachen Debugging Ihres Programms. Benutzen Sie die Ihnen bereits bekannte `_delay_ms()`-Funktion zum Einhalten der Wartezeiten.

Hausaufgabe:
Um was für eine Art von Flip-Flops handelt es sich im Schieberegister? Zustandsgesteuerte? Taktflankengesteuerte? Haben wir es mit *Latches* zu tun? Begründen Sie Ihre Antwort und versuchen Sie die Antworten zum einen aus der Literatur (z. B. Datenblatt, aber auch Ergänzungsliteratur) *und* zusätzlich auch experimentell zu bestimmen.

Gerne dürfen Sie sich an einem Lauflicht versuchen oder an einer Visualisierung eines langsam hochzählenden binären Zählers (der von 0 bis einschließlich 7 zählt und das Bitmuster jedes Zählerstandes über die Leuchtdioden ausgibt).

Diese Hausausgabe können Sie auch im *virtuellen Labor* bearbeiten. Dort können Sie über das Menü das Programm `IES-SimulIDE` starten. In diesem Programm können Sie dann die Datei `Schieberegister.simu` öffnen, die Sie im moodle im Bereich `SimulIDE-Schaltungen` finden. Nach dem öffnen der Datei finden Sie eine einfache Schieberegisterschaltung, die an einen simulierten ATmega328-Microcontroller angeschlossen ist, die die gemäß der Beschaltung den Gegebenheiten am Roboter entspricht, d. h., dass die Beinchen genau so wie dort beschaltet sind.

Die Datei, die Sie per `avrdude` auf den Roboter flashen würden, können Sie per Kontextmenü des simulierten Microcontrollers öffnen. Auf diese Art wird die HEX-Datei in den simulierten Microcontroller gelesen. Danach können Sie die Simulation starten. Ihr Microcontroller-Programm wird sich wie auf der echten Hardware verhalten.

Aufgabe 13: Abschließendes für diese Übung
Kombinieren Sie alles, was Sie heute gelernt haben. Entwickeln Sie ein Programm, dass die Linienfolgerzustände (digital, nicht analog . . . auch dann nicht, falls Sie bereits wissen, wie das geht) ausliest.
Sollte ein Linienfolger kein reflektiertes Licht detektieren (sich beispielsweise über einer schwarzen Linie befinden), lassen Sie die an das Schieberegister angeschlossene passende Leuchtdiode leuchten. *Passend* bedeutet hier: Sollte der in Roboter-Fahrtrichtung linke Linienfolger eine schwarze Linie detektieren, soll die linke (blaue) Leuchtdiode leuchten. Entsprechend gehört zum mittleren Linienfolger die mittlere (grüne) Leuchtdiode und zum rechten Linienfolger passt die rechte (gelbe) Leuchtdiode.

Geben Sie außerdem per serieller Schnittstelle bei jeder Änderung (aber nur bei/direkt nach einer Änderung) aus, welcher Linienfolger gerade eine schwarze Linie erkennt (einer, zwei, drei, keiner).

Generelle bei Fehlern zu bedenkende Dinge:
Auftretende Fehler, die Ihnen am Anfang vielleicht das Leben in diesem Praktikum schwer machen sind beispielsweise:

- Kabel nicht angeschlossen
- Cutecom-Verbindung beim Flashen noch geöffnet
- (Später) Bluetooth-Fähnchen noch gesteckt
- Executable-Bit vergessen bei `compile.sh`
- Fehlenbde oder falsche Include-Angaben
- Mitkompilieren einer C-Datei vergessen (z. B. `iesusart.c`)
- Endlosschleife vergessen
- Datenrichtungs-Einstellungen (DDR-Register) vergessen oder falsche Richtung gewählt