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 2

Schieberegister, Simulation, Antriebsstrang

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

Aufgabe 0: Wichtig!

Legen Sie die Akkus NOCH NICHT in den Roboter ein! Tun Sie das erst, wenn Ihnen das vom Betreuer gestattet wurde!

Aufgabe 0b: Auch wichtig!

Man kann keine Ausdrücke, die nur Nullen enthalten, sinnvoll shiften, um eine einzelne Bitstelle zu löschen!

Überlegen Sie sehr genau, was beispielsweise die Anweisung `x = 0 << 2;` bedeutet; hier wird der Variable `x` immer (ausschließlich!) der Wert `0` zugewiesen. Warum ist das so? Rechts vom Gleichheitszeichen finden wir den Ausdruck `0 << 2`. Hier wird dann relevant, was links vom Shiftoperator geschieht: Es wird ein Wert `0` (mit dem entsprechenden Bitmuster) angelegt; d. h., dass alle Bits, die geshiftet werden sollen, auf 0 gesetzt sind. Was aber passiert, wenn man lauter Nullen um eine beliebige Zahl shiftet? Es ändert sich gar nichts, der Gesamtausdruck bleibt **0**.

Es sind nur Werte ungleich `0` sinnvoll shiftbar. Zum Löschen einer Bitstelle kommen Sie nicht darum herum, ein Bit ganz explizit mittels eines Ausdrucks (Stichwort: Bitmaske!), der unter anderem die bitweise Negation und den Konjunktionsoperator enthält,zu löschen!

Aufgabe 0b: Außerdem wichtig!

Melden Sie sich zur Prüfung an!

Aufgabe 1a: Das Schieberegister – Ausführliche Erklärung und Beispielcode-Besprechung

Eventuell haben Sie die letzte Afgabe auf Übungsblatt 1 bisher nicht vollständig bearbeiten können.

Im moodle finden Sie einen C-Quelltext mit dem Titel `sreg.c`. Dieser Quelltext stellt dar, was beispielsweise Ergebnis der letzten Aufgabe des ersten Übungsblatts hätte sein können. Testen Sie vorab die Programmfunktionalität des erwähnten Quelltexts, kompilieren und flashen Sie ihn also, testen sie ihn dann mit dem **T**rackstück. Sie haben insgesamt fünf Minuten Zeit. Melden Sie sich, wenn Sie vorher fertig sind.

Den Quelltext werden wir nun gemeinsam durcharbeiten. Öffnen Sie ihn in geany. Halten Sie dieses Übungsblatt, insbesondere Abb. 1 in einem weiteren Fenster geöffnet und sorgen Sie dafür, dass Sie während des vom Betreuer angeleiteten Durcharbeitens *kognitiv dabei bleiben* (körperlich sowieso). Melden Sie sich notfalls.

In Abb. 1 finden Sie ein konzeptuelles Schaltbild eines 3-Bit-Schieberegisters. Die gestrichelte Linie soll ein IC-Gehäuse darstellen. Am IC-Gehäuse des Modells des auf den Robotern verbauten Schieberegisterbausteins hängen natürlich viel mehr Beinchen, als im Schaubild dargestellt sind. Tatsächlich enthält der verbaute Baustein sogar *zwei* Schieberegister, von denen aber nur eines angeschlossen ist. Manche Beinchen des Bausteins sind also nirgends angeschlossen und sind *blind* auf der Platine verlötet. Wir interessieren uns hier vorerst also nur für eines der beiden Schieberegister und insgesamt fünf Beinchen. Das Schieberegister, das angeschlossen

ist, verfügt über einen Dateneingang (hängt an Beinchen `PB2`), einen Takteingang (`PD4`) und drei Ausgänge. Die drei Ausgänge gehören zu den drei ersten Flip-Flops, aus denen das angeschlossene Schieberegister aufgebaut ist (eigentlich besteht es aus vier Flip-Flops, es handelt sich also um ein 4-Bit-Schieberegister – das vierte Flip-Flop ist jedoch nicht angeschlossen.).

Die Flip-Flops in einem Schieberegister sind in einer Reihe miteinander verbunden. Nur der Dateneingang des ersten Flip-Flops ist nach außen geführt (vgl. Schaubild). Der Ausgang des Schieberegisters ist an den Eingang eines weiteren Schieberegisters angeschlossen. Außerdem wird dieser Ausgang auch nach außen gefühjrt. Auf dem Roboter hängt an diesem Ausgang über einen Vorwiderstand die blaue Leuchtdiode. Der Ausgang des zweiten Flip-Flops ist wiederum an den Eingang eines weiteren, dritten Flip-Flops angeschlossen. Außerdem hängt an dem Ausgang des zweiten Flip-Flops nicht nur der Eingang des dritten Flip-Flops, sondern auch die grüne Leuchtdiode. Der Ausgang des dritten Flip-Flops ist wiederum an ein weiteres Flip-Flop und an die gelbe Leuchtdiode angeschlossen.

Das IC-Gehäuse enthält einen nach außen geführten Takteingang, der an alle Takteingänge der D-Flip-Flops im IC-Gehäuse angeschlossen ist. Wenn der Takteingang einmal geschaltet (eine steigende Flanke erzeugt) wird, übernimmt das erste Flip-Flop den Wert (`0` oder `1`), der gerade am Dateneingang des IC-Gehäuses anliegt, an seinen Ausgang. Dann leuchtet die blaue LEuchtdiode. Da am Ausgang des ersten Flip-Flops – Q_1 – nun eine `1` anliegt, übernimmt das zweite Flip-Flop mit der nächsten steigenden Flanke diesen Wert an Q_2. Das selbe gilt dann auch für das dritte Flip-Flop un Q_3, mit der nächsten steigenden Taktflanke.

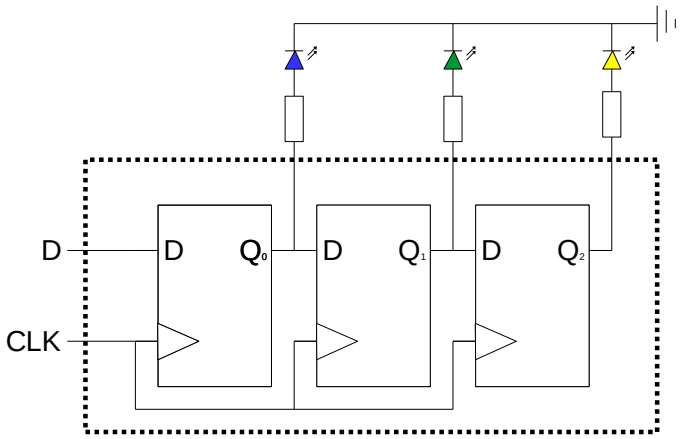

Abbildung 1: 3-Bit-Schieberegister mit D-Flip-Flops

Natürlich gilt auch: Bei den diversen Taktflanken wird in das erste Flip-Flop bzw. an Q_1 stets das übernommen, was am IC-Dateneingang anliegt.

Das bedeutet auch: Wenn man explizit (nur!) die gelbe Leuchtdiode leuchten lassen will, dann muss man bei einer ersten Taktflanke eine `1` auf den Dateneingang schalten und dann einmal eine steigende Flank am Takteingang erzeugen. Das Erzeugen einer steigenden Flanke kann (muss, in diesem Fall hier) dadurch geschehen, dass man das am Takteingang angeschlossene Microcontrollerbeinchen, das natürlich als Ausgang geschaltet sein muss, von 0 auf 1 schaltet. Nachdem erstmalig eine steigende Flanke erzeugt wurde, muss der Dateneingang des Schieberegisters wieder auf `0` gesetzt werden. Danach muss noch zwei Mal eine steigende Flanke am Takteingang erzeugt werden, damit die initial eingespeiste `1` durch das Schieberegister *geschoben* wird. Dann wird die dritte Leuchtdiode leuchten.

Behalten Sie das im Hinterkopf, wenn der Betreuer mit Ihnen nun `sreg.c` durchsprechen wird.

In dem Quelltext werden Ihnen eine Menge neuer (primär: Quelltextgestaltung-) Konzepte begegnen, die zumindest zum Teil in Ihrer Abschlussaufgabe explizit gefordert werden. Also passen Sie gut auf. Sehen Sie die Konzepte als dringende Vorschläge. Sehen Sie den Quelltext jedoch nicht als Musterlösung, sondern als Lösungsvorschlag, den Ihr Betreuer (erstmal) ganz gut findet.

Aufgabe 1b: Simulation des Roboters und entsprechender Algorithmik

Der Betreuer wird das hier vorturnen. Denken Sie daran, dass das virtuelle Labor auch im moodle zum Download bereit steht, Sie können alle Schritte also auch daheim nachspielen.

Laden Sie die Datei `simulide` im moodle. Kopieren Sie diese Datei dann aus beispielsweise Ihrem Home-Verzeichnis in ein (verstecktes) Unterverzeichnis Ihres Home-Verzeichnisses, das `.bin` heißt (und bereits existiert). Geben Sie, nach dem Download, in einem Terminal den folgenden Befehl ein:

```
cp simulide  /.bin/ies_simulide/
```

Öffnen Sie danach im virtuellen Labor über das Anwendungs-Menü im Untermenü `Education` das Simulationswerkzeug `IES-SimulIDE` (nicht *SimulIDE*). Bei diesem Tool handelt es sich um ein simulationstechnisches Großkunstwerk, zu dem Sie unter `https://www.simulide.com` (Datum des letzten Downloads: 15. Juni 2022) viele Informationen finden. Natürlich handelt es sich um ein *Open-Source-Werkzeug* gemäß der *GPL v3*-Lizenz (vgl. `https://www.gnu.org/licenses/gpl-3.0.en.html` ; Datum des letzten Downloads: 15. Juni 2022).

Mit SimulIDE ist es möglich, kleine elektronische Schaltungen, die als Peripherie-Beschaltungen für viele unterschiedliche Microcontroller (der Microchip AVR- und PIC-Reihe) dienen, simulativ aufzubauen und sie dann mit einem Programm, das vom Microcontroller abgearbeitet wird, zu bespielen.

Bei `IES-SimulIDE` handelt es sich um ein *aufgebohrtes* SimulIDE. Am FG IES ist derzeit eine Erweiterung für SimulIDE in Entwicklung, die es erlaubt, den Roboter zu simulieren. Die roboter-Simulation befindet sich derzeit noch in einem Alpha-Stadium, ist aber bereits partiell nutzbar. Derzeit ist die Erweiterung noch nicht in SimulIDE contributed, was aber – sobald ein ausreichend großer Funktionsumfang vorhanden ist und noch bestehende Fehler behoben sind – geplant ist. Halten Sie hierzu ggf. nach am FG IES ausgeschriebenen Projektarbeiten Ausschau.[1]

Nachdem IES-SimulIDE gestartet wurde, sehen Sie eine vermeintlich leere Arbeitsoberfläche (Workbench) und einige Menüs. Tatsächlich wird beim Start von IES-SimulIDE jedoch bereits eine Schaltung mit geladen. Klicken Sie mit der rechten Maustaste auf die Workbench und wählen Sie im sich öffnenden Kontextmenü `Zoom to fit`. Nun füllt eine einfache Schaltung, die aus einem Microcontroller, einigen Widerständen, Leuchtdioden und einem Schieberegister besteht. Das Schieberegister sieht etwas komplexer aus als das, das Ihnen bereits konzeptuell erklärt wurde. Sie müssen das in der simulierten Schaltung verbaute Schieberegister bzw. die dort zusätzlich vorhandenen Anschlüsse/Beschaltungen nicht verstehen. Aber denken Sie im Rahmen einer **Hausaufgabe** darüber nach, was diese Anschlüsse bedeuten und wofür man sie einsetzen kann.

Durch einen Rechtsklick auf den Microcontroller öffnet sich ein Kontextmenü, in dem Sie `Load firmware` auswählen. Im sich dann öffnenden File-Requester wählen Sie die HEX-Datei, die Sie beim Compilieren beispielsweise des Schieberegister-Beispielcodes erzeugt haben; zur Erinnerung: Das ist die Datei, die von `avr-objcopy` erzeugt und dann von `avrdude` auf den *echten* Microcontroller geflasht würde. Beachten Sie, dass Sie in Ihrem Compile-Script ggf. den Aufruf zum Löschen des Tempfiles entfernen müssen. Dabei handelt es sich um die Zeile, die mit `rm` beginnt.

Wählen Sie dann per Kontextmenü des großen schwarzen Klotzes den Eintrag `Start PyGame`. Im sich öffnenden Fenster sehen Sie ein kleines Auto von oben, das auf einem einfachen Linientrack steht.

Öffnen Sie auf der Workbench in IES-SimulIDE den seriellen Port. Die Datei `dev/ttyp0` steht im virtuellen Labor als serieller Endpunkt einer (per `socat` realisierten) virtuellen seriellen Verbindung zur Verfügung und sie wird durch den Workbench-Button geöffnet. Als weiterer Endpunkt steht auch die Datei `dev/ttyp1` zur Verfügung, die Sie in Cutecom (im Dropdown-Menü innerhalb cutecoms) öffnen können. Danach können Sie mit dem simulierten Microcontroller innerhalb kommunziieren, wie Sie es auch mit dem echten Micrcontroller tun. Sollte `/dev/ttyp1` in cutecom nicht zur Verfügung stehen, schließen Sie cutecom und öffnen Sie es erneut. Aufgrund eines Fehlers in einer Qt-Bibliothek erkennt cutecom nicht immer alle im System vorhandenen seriellen Ports.

Nachdem Sie `PyGame` und cutecom nun also geöffnet haben, starten Sie die Simulation in IES-SimulIDE, durch einen Klick auf den großen roten Anschaltknopf über der Workbench. Nun beginnt der simulkierte Microcontroller, Ihr Programm abzuarbeiten.

Im PyGame-Fenster können Sie den Roboter verschieben, und zwar dadurch, dass Sie ihn mit der rechten Maustaste anklicken und die Maus bei gedrückter rechter Maustaste bewegen. Bewegen Sie den Roboter mit den Linienfolger-Elementen über einer schwarzen Linie hin und her, und schauen Sie, was in der Workbench (Leuchtdioden) und im cutecom-Fenster (Roboter-Nachrichten) geschieht. Spielen Sie rum!

[1] Werbung ein eigener Sache. ;-)

Aufgabe 2: Der Antriebsstrang - Grundlagen

Legen Sie die Akkus NOCH NICHT in den Roboter ein!

Ein Motor braucht eine gewisse Mindestspannung, damit er überhaupt anläuft.

Ab dieser Mindestspannung kann man die Spannung variieren, so dass der Motor unterschiedlich schnell dreht.

Nehmen wir mal – nur für den Moment – an, dass wir Motoren direkt an ein Beinchen unseres Microcontrollers anschließen *könnten* (was wir aber in der Praxis keinesfalls dürfen!), dann könnten wir durch ein Schalten des Beinchens auf 0 den Anschluss des Motors gegen Masse (`GND`) ziehen, bei Schalten des Beinchens auf 1 würden wir den Anschluss des Motors gegen die Versorgungsspannung des Microcontrollers (`VCC`) ziehen.

Das heißt, wir könnten unseren Motor nun entweder mit $0V$ oder (knapp) $5V$ betreiben.

Das würde bedeuten, dass unsere Motor entweder gar nicht oder mit einer bestimmten stets festgelegten Geschwindigkeit drehen würde.

Was wäre aber, wenn unsere Motoren sowieso (viel) mehr Spannung bräuchten als die `VCC` des Microcontrollers?

Hier kommt der L298N-Brückenbaustein ins Spiel.

Die vier auf dem Roboter befindlichen Motoren sind über den Brückenbaustein an den Microcontroller angeschlossen, d. h.:

Die vier gelben Motoren sind am Brückenbaustein eingesteckt. Der Brückenbaustein selbst ist mit (insgesamt sechs) Microcontrollerbeinchen verbunden.

Die in Fahrtrichtung *rechten* Motoren sind an die Brückenbausteinausgänge `OUT1` und `OUT2` angeschlossen, die in Fahrtrichtung *linken* Motoren sind an die Ausgänge `OUT3` und `OUT4` angeschlossen.

Die Beinchen des Microcontrollers sind an den Brückeneingängen `ENA`, `ENB`, `IN1`, `IN2`, `IN3` und `IN4` angeschlossen.

Abhängig davon, wie man diese Beinchen schaltet/ansteuert, werden die Motoren dann vom Brückenbaustein angesteuert.

Durch belegen der Eingänge `IN1` und `IN2` mit jeweils entweder 0 oder 1 steuert man die Richtung des linken Motors.

Durch belegen der Eingänge `IN3` und `IN4` mit jeweils entweder 0 oder 1 steuert man die Richtung des linken Motors.

Durch das Beschalten der Eingänge `IN1`, `IN2`, `IN3` und `IN4` gibt man jedoch nur die *Drehrichtung* vor. Dass wirklich Motoren angetrieben werden sollen, entscheidet man über die Eingänge `ENA` und `ENB`.

Sollte der *linke* Motor auf die Drehrichtung *vorwärts* gestellt sein, so kann man danach (oder auch davor) den Eingang `ENA` mit einer 1 ansteuern. Dann beginnen die linken Motoren zu drehen. Mit `ENB` schaltet man die *rechten* Motoren an.

Legen Sie die Akkus NOCH NICHT in den Roboter ein!

Aufgabe 3: Die Motoren anschalten

Legen Sie die Akkus NOCH NICHT in den Roboter ein!

Laden Sie aus dem moodle-Kurs die Dateien `skeleton_motors.c`, `iesmotors.h` und `iesmotors.c`.

Setzen Sie Ihren Roboter auf einen Holzbock!

Kompilieren, transformieren und flashen Sie den Code aus den o. a. Dateien per USB-Kabel auf Ihren Roboter.

Legen Sie ERST JETZT die Akkus in den Roboter ein, achten Sie dabei unbedingt auf die Polarität!

Schalten Sie jetzt Ihren Roboter ein (per Schiebeschalter am Akkuträger).

Denken Sie an den Holzbock unter Ihrem Roboter! Wann auch immer Sie Akkus in den Roboter legen, dürfen die Räder keinen Bodenkontakt haben, es sei denn, der Roboter steht im Track-Rahmen!

Ihr Roboter wird beginnen, die Motoren in unterschiedliche Richtungen zu bewegen. Nachvollziehen Sie den Code, stimmen Sie ihn mit dem ab, was der Roboter tut. Verstehen Sie also das Verhalten Ihres Roboters und die einzelnen Schritte im Quelltext.

Für später, ggf. als **Hausaufgabe**, auf jeden Fall aber zur komplett selbstständigen Arbeit:

Erzeugen Sie eine Wahrheitstabelle, die abhängig von der Belegung (Beschaltung) der Brückeneingänge angibt, welcher Motor/welche Motorenseite bei welcher Belegung was tut. Die Tabelle könnte als Spaltenüberschriften beispielsweise alle Anschlüsse der H-Brücke enthalten und eine Spaltenüberschrift zur Roboteraktion (gerade aus fahren, links lenken, rechts lenken) und eine Spaltenüberschrift zur Fahrtrichtung (vorwärts, rückwärts) enthalten.

Überlege Sie außerdem konzeptuell, wie Sie Dreh- bzw. Lenkbewegungen mit Ihrem Roboter durchführen können. Wie müssten die Motoren angesteuert werden, um ein Lenken nach rechts oder links umzusetzen?

Für später, ggf. als **Hausaufgabe**: Räumen sie den Code auf. Ersetzen Sie die „Beinchenkonstanten“ aus der AVR-Bibliothek durch sinnvolle, zum Roboter passende Konstantennamen. Verwenden Sie dafür Präprozessoranweisungen. Sie können sich dazu beispielsweise an dem Vorschlag zur Schieberegister-Ansteuerung orientieren, in dem *Roboter-Funktions-bezogene* Konstanten eingeführt wurden. Ihr Code muss sehr leicht lesbar sein, und zwar derart, dass kein Datenblatt und kein Pinmapping-Dokument benötigt um die konzeptuelle Funktion Ihres Codes schnell zu verstehen.

Schalten Sie Ihren Roboter aus, sobald Sie mit der Bearbeitung dieser Aufgabe fertig sind. Schalten Sie ihn erst wieder an, nachdem Sie ihn – später – auf den Linientrack gesetzt haben.

Aufgabe 4: Die Geschwindigkeit der Motoren steuern und Pulsweitenmodulation

Der Brückenbaustein schaltet an die Motoren eine andere Spannung als die `VCC` des Microcontrolers durch. Bei der durchgeschalteten Spannung handelt es sich quasi um die als VIN bezeichnete *Eingangs*spannung des Microcontrollers. Auf dem Arduino Uno-Board arbeitet ein Spannungsregler, der eine „höhere“ Eingangsspannung auf (ungefähr) $5V$ (`VCC`) herunterregelt. Diese Eingangsspannung wird von den Akkus bereit gestellt und je nach Ladezustand rangiert die „irgendwo“ zwischen $3,3V$ und $4,2V$ pro Akku, die meiste Zeit im Bereich von $3,7V$. Da wir zwei Akkus verbaut haben und diese *in Reihe* verschaltet sind, haben wir es (meistens) mit einer `VIN` von $7,4V$ zu tun.

Nochmal aus der Anleitung zu den Akkus:
Die Ladung, bis zu der ein Akku entladen werden kann, nennt sich Entladeschlussspannung. *Sobald diese bei den von uns verwendeten Lithium-Ionen-Akkus unterschritten wird – der Akku also tiefentladen wird – werden die Akkus unwiederbringlich zerstört. Beim Laden eines maximal tiefentladenen Akkus kann dieser sogar abbrennen/explodieren. Um eine Tiefentladung zu vermeiden, verfügen die von uns verwendeten Lithium-Ionen-Akkus über eine Schutzschaltung. Diese befindet sich im Akku unter dem Pluspol. Sie verhindert eine Tiefentladung und eine Überladung. Dennoch müssen Sie achtsam mit den Batterien umgehen. Es ist gesünder, die Batterien möglichst oft zu laden, als sie häufig zu entleeren. Sie sollten Ihre Akkus spätestens nach insgesamt 10 Minuten Motornutzung (Fahrzeit) wieder laden!*

Über den Brückenbaustein steht also die (veränderliche) `VIN` den Motoren zur Verfügung. Abhängig von der Höhe von `VIN` drehen die Motoren unterschiedlich schnell. Je länger die Motoren laufen – ein Roboter kann etwa 45 Minuten im Kreis fahren, mit einer vollständigen Ladung beider Akkus, er wird dabei jedoch immer träger – desto langsamer drehen die Motoren. Sobald die Spannung an einem Akku ungefähr $3,3V$ unterschreitet, bringen die Akkus keine Leistung mehr (denken Sie an $P = U \cdot I$ ... was war das nochmal?). Ganz frisch aufgeladene Akkus drehen die Motoren also am schnellsten, „fast leere“ am langsamsten. Die Schutzschaltung in den Akkus verhindert eine Tiefentladung – testen Sie das jedoch nicht!

Wenn wir die Brückenbausteineingänge `ENA` und `ENB` dauerhaft auf 1 schalten (durch die Microcontrollerbeinchen `PD5` und `PD6`), dann wird solange wie sie auf 1 gesetzt sind, die volle zur Verfügung stehende VIN (die mit der Zeit jedoch abnimmt) an die Motoren geleitet.

Um *gezielt weniger* Spannung (also weniger als VIN) an die Motoren zu leiten, muss man hier zu einem Trick greifen:

Statt `ENA` oder `ENB` *sehr lange* auf 1 geschaltet zu lassen, kann man auch zwischen 1 und 0 hin und her schalten.

Wenn innerhalb einer Zeitperiode T für eine Zeitdauer D mit $D <= T$ verinbart wird, dass ein Signal (das z. B. vom Microcontrollerbeinchen erzeugt wird) während der Dauer D den Wert 1 haben soll, sonst aber den Wert 0, dann entspricht die Fläche unter diesem Signalabschnitt (also während der Periode T) dem Wert $D \cdot 1$. Wenn $D = T$, dann entspricht die Fläche unter diesem Signalabschnitt während der Periode T dem Produkt aus Periode und Signalwert selbst.

Das Verhältnis von D zu T, also $\frac{D}{T}$, kann man damit als den Anteil der *effektiv* während der Periode T übertragenen Spannung interpretieren (wir erinnern uns: die Logische 1 am Microcontrollerbeinchen entspricht dem Ziehen des Beinchens nach `VCC`). Falls $T = 2D$, dann wäre das Verhältnis also $0,5$ (oder: 50%) und wir hätten (über die einzelne Periode betrachtet oder auch über alle Perioden – solange sich das Verhältnis $\frac{D}{T}$ nicht ändert) also effektiv $2,5V$ im Signal übertragen.

Diese Art ein Signal zu *modulieren* nennt man *Pulsweitenmodulation*.

Wenn wir unser Signal nun *periodisch* mit der Periode(ndauer) T anlegen bedeutet das, dass immer ab 0 beginnend alle T Zeitschritte ein Microcontrollerbeinchen auf 1 gesetzt wird, und zwar jeweils für die Dauer D.

Wenn T hinreichend kurz gehalten wird (also: sehr kurz), die Frequenz mit der die Schaltvorgänge durchgeführt werden also (sehr) hoch ist, bemerkt man fast nie, dass wir kein *kontinuierliches* Signal haben (mit dem wir quasi „jeden“ Spannungswert zwischen 0 und `VCC` darstellen können), sondern immernoch ein *digitales*.

In der Fachliteratur (oder auch in Handbüchern und Datenblättern) wird D üblicherweise mit *Duty-cycle* oder *Tastgrad* oder *Tastrate* beschrieben.

In Abbildung 2 sind einige Signalverläufe über jeweils vier Perioden mit unterschiedlichen Werten für D dargestellt.

Falls wir nun ein T und ein entsprechendes$D$ bestimmen, könnten wir die Beinchen an denen `ENA` und `ENB` angeschlossen sind, per Software hin und her schalten und dadurch den Brückenbaustein dazu bringen, die Motoren mit einem dem vom Microcontrollerbeinchen vorgegebenen Duty-Cycle mit `VIN` zu beschalten. Dadurch würden die Motoren dann, abhängig vom von uns gewählten Duty-Cycle, unterschiedlich schnell laufen.

Zum technischen Prinzip: Auf dem Brückenbaustein wird quasi „sowas wie“ ein Transistor durchgeschaltet. Die Motoren sind in die Kollektor-Emitter-Strecke gehangen, `EN[A|B]` entspricht der Transistorbasis.

Eine solche Pulsweitenmodulation *kann* man per Software umsetzen, und das ist auch, je nach Anwendung, nicht vollkommen ungewöhnlich. Üblicherweise lässt man das jedoch die Microcontroller-Hardware erledigen, indem man diese ein mal einstellt und dann „automatisch“ die Beinchen gemäß eines gewählten Duty-Cycles „pulsen“ lässt.

Dazu nutzt man einen auf dem Microcontroller verbauten Timer. Dabei handelt es sich um ein (über einen einstellbaren Taktvorteiler) an den Taktgeber angeschlossenes Register, dessen Inhalt mit jedem Takt inkrementiert wird. Der ATmega328p hat insgesamt drei Timer, wobei uns hier nur *Timer 0* interessiert. Bei diesem Timer handelt es sich um einen *8-Bit-Timer*.

Es wird, falls keine Taktvorteilung eingestellt ist, mit jeder Taktflanke von 0 beginnend um 1 hochgezählt. Wie Sie das auch von (ganzzahligen) Variablen in C kennen, kommt es bei $255 + 1$ zu einem Überlauf und der Timer startet wieder von vorne. Je nach dem, wie Timer 0 (in den Einstellungsregistern `TCCR0[A|B]` (`TCCR` steht für *Timer/Counter Controll Register*)) eingestellt ist, zählt er einfach hoch oder er zählt hoch bis sein Zählerstand dem Inhalt eines der beiden beiden Register `OC0[A|B]` entspricht. Sobald der Timerwert und der Wert in einem der beiden `OC0`-Register gleich sind, wird das von einem im Microcontroller verbauten *Waveform Generation Module (WGM)* registriert. Wenn das der Fall ist, übernimmt das WGM das Umschalten von 1 auf 0 an einem der Pins, die im TCCR0A definiert wurden (dort stellt man ein, welche Pins – es sind aber nur PD5 und PD6 möglich) umgestellt werden sollen.

Die Brückenbausteineingänge `EN[A|B]` hängen an den Microcontrollerbein `PD5` und `PD6`. Schlagen Sie das Microcontrollerdatenblatt auf Seite 14 erneut auf. Dort stehen die Bezeichnungen der verschiedenen Beinchen und auch die *Alternativfunktionen* der Beinchen, falls es sich bei den Beinchen um solche Beinchen handelt, die Sie statt als Ausgabe- oder Eingabebeinchen auch noch als Beinchen „für etwas Anderes“ nutzen können. Sie sehen bei beiden Beinchen z. B. die Bezeichnung `OC`, was für *Output Compare* steht.

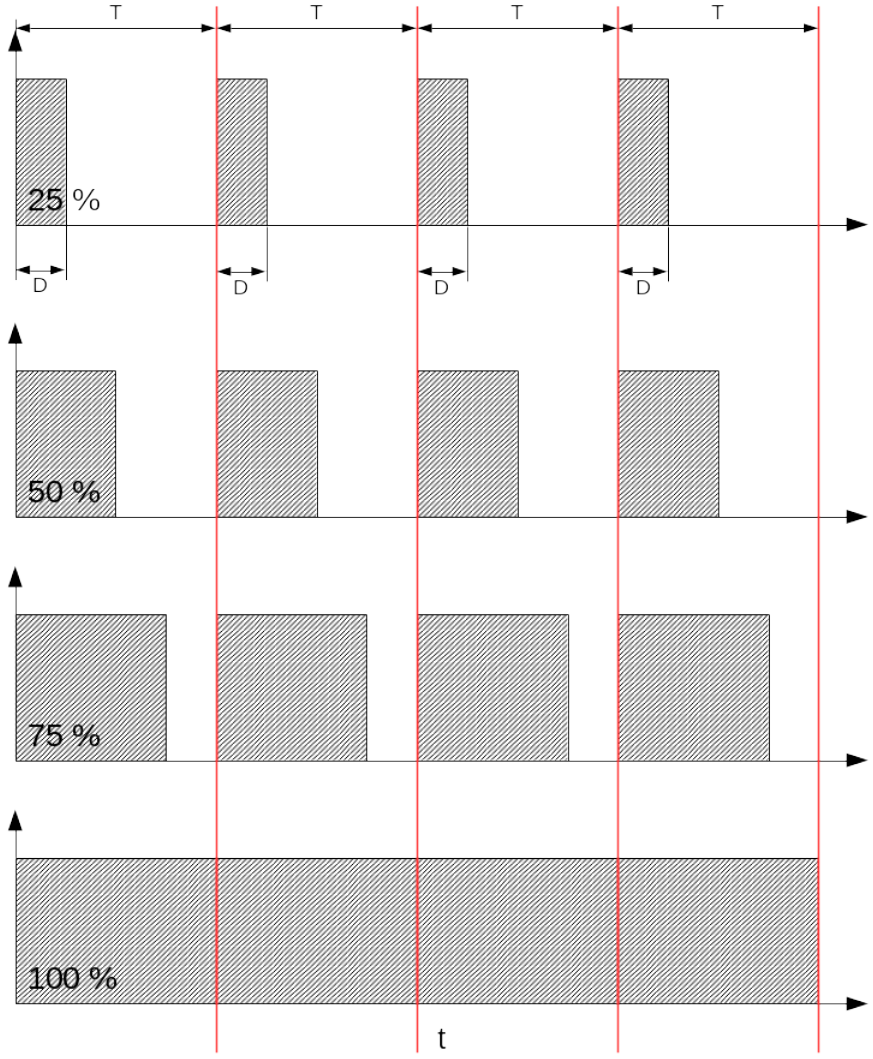

Abbildung 2: Verschiedene Duty-Cycles

In der Datei `iesmotors.c` ist Ihnen das Einstellen des Timer 0 und auch das setzen eines Duty-Cycles bereits vorgegeben. Sie brauchen die Funktion nur zu benutzen. Verstehen Sie aber unbedingt – vielleicht werden Sie das ja im Rahmen der Abschlusspräsentation – gefragt, worum es sich dabei handelt; im Rahmen der Abschlusspräsentation wird verifiziert werden, ob Sie Ihren abgegebenen Code selbst geschrieben und verstanden haben.

Wichtig:

1. *Bevor* Sie die Beinchen PD5 und PD6 als Ausgänge (z. B. für PWM an ENA und ENB) nutzen können, müssen sie diese Beinchen unbedingt im passenden DDR auf „Ausgang“ setzen – stellen Sie in Ihrem Code explizit sicher, dass das der Fall ist.
2. Führen Sie dann die Funktion `setupTimer0(void)` aus.
3. Nutzen Sie dann die Funktion `setDutyCycle(uint8_t pin, uint8_t value)` um anzuweisen, welcher Pin (PD5 oder PD6, nutzen Sie diese oder von Ihnen definierte Konstanten) mit welchem Duty-Cycle-Wert (*value*) betrieben werden soll.

Hausaufgabe: In dieser Aufgabe und auch im Quelltext begegnen Ihnen die Konstanten `P`*`kx`* statt `PORT`*`kx`*. Warum kann man sie verwenden, woher stamme sie? Recherchieren Sie dafür die Header-Dateien `avr/io.h` und `portpins.h`. Schauen Sie besonders, was in Verbindung mit diesen Konstanten ind `portpins.h` geschieht (und warum).

Aufgabe 5: Programmieren - Pflicht für alle

Wichtig: Das erstmalige Aufsetzen Ihres roboters auf den Linientrack machen Sie a) erst, wenn Ihre in dieser Aufgabe zu entwickelnde Algorithmik halbwegs passabel auf dem Holzblock mit dem Trackstück getestet funktioniert und b) in Begleitung eines Hiwis!

Jetzt geht es darum, dass Sie Ihren Roboter fahren lassen, und zwar die schwarze Linie entlang.

Entwickeln Sie ein Programm, dass fortlaufend prüft, ob Ihr Roboter noch auf dem richtigen Weg ist.

Sie wissen ja: Wenn (nur) das mittlere Linienfolgerelement „keine Reflektion“ meldet, dann sind Sie damit auf der schwarzen Linie unterwegs. Das bedeutet auch, dass Sie geradeaus fahren können.

Leider ist es so, dass alle verbauten Sensoren des Roboters ziemlich „streuen“, also sie sind nicht sonderlich genau. Auch die Aktorik (also die Motoren) arbeitet unterschiedlich. Vor allem aber gibt es (immer und grundsätzlich bei jedem technischen Produkt) gewisse Fertigungstoleranzen und irreduzible Ungenauigkeiten. Z. B. kann es sein, dass Ihr Roboter eine Drift nach links oder rechts hat. Das ist *vollkommen normal* und eine Ihrer Aufgaben im Praktikum ist es, das meiste *aus Ihrem* Roboter heraus zu holen, die technischen Probleme z. B. per Software zu lösen.

Das bedeutet aber auch, dass selbst auf einer schnurgeraden Strecke kein perfekter Geradeauslauf möglich ist. Das bedeutet wiederum, dass Sie eine Algorithmik entwickeln müssen, die die Richtung des roboter steuert. Also: Sie müssen beim Fahren bitte lenken.

Sie wissen hierzu z. B. auch, dass Sie Ihren Roboter nach links steuern müssen, falls das linke Linienfolgermodul „keine Reflektion“ meldet und entsprechend nach rechts lenken, falls das rechte Linienfolgermodul feuert. Das Gegenlenken könnten Sie z. B. so lange durchführen, bis der mittlere Linienfolger wieder feuert.

Spielen und experimentieren Sie.

Jeder Ihrer Roboter wurde geprüft und fuhr den Linientrack bereits (viele Male) ab. Solange Ihr Roboter das Programm, das in `skeleton_motors.c` notiert ist, ausführen kann, ist alles in Ordnung. Testen Sie Ihre Algorithmik vor dem Aufsetzen auf den Linientrack ausgiebig auf dem Holzbock.

Lernen Sie Ihren Roboter nun kennen. Viel Spaß!

Aufgabe 6: Programmieren – Freiwillig für Spielbegeisterte im Rahmen einer *Hausaufgabe*

Entwickeln Sie eine PWM-Methode um eine der am Roboter verbauten Leuchtdioden – es bietet sich die auf dem Arduino-Shield mit `L` bezeichnete Leuchtdiode an, und Sie wissen bereits, an welchem Microcontrollerbeinchen diese angeschlossen ist – zu dimmen, also sie unterschiedlich hell leuchten zu lassen. Hierzu dürfen Sie aber keinen Timer verwenden, sondern Sie müssen das „zu Fuß“, also in Software erledigen.

Sie benötigen dazu nichts weiter als eine der Wartefunktionen und ein bisschen (wirklich nicht viel) programmiertechnisches Geschick.

Erweitern Sie Ihr Programm dann abschließend derart, dass Sie aus einem seriellen Terminal am Rechner heraus Werte für den Duty-Cycle, den Sie an die Leuchtdiode senden wollen, an den Microcontroller übertragen. Die Leuchtdiode soll dann einen entsprechenden Helligkeitswert annehmen.

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

- Sind die Akkus richtig eingesetzt, also: Wurde die Polarität beachtet? Das flache Akku-Ende ist der Minus-Pol und er muss mit der im Akkuträger verbauten Feder Kontakt haben.
- Falls der Roboter nicht mehr am USB-Kabel hängt: Wurde der Roboter per Schiebeschalter angeschaltet?
- Sind die Akkus voll (genug) geladen? Laden Sie nach etwa 10 Minuten reiner Fahr-Zeit.
- Zu bedenken ist, dass die Akkuspannung relativ schnell nachlässt, ausgehend von 4,2 Volt mit steiler Entladekennlinie bis sie sich bei ungefähr 3,7 Volt bis 3,9 Volt deutlich abflacht. Das werden Sie am Verhalten Ihre roboters merken! Arbeiten Sie erstmal am besten mit voll geladenen Akkus. Später werden Sie die Möglichkeit haben, die an den Akkus anliegende Spannung (grob) zu messen und beispielsweise Ihre Duty-Cycles an den Ladezustand der Akkus anzupassen. (Das geht jetzt schon in Richtung Bonuspunkte bei der Abschlussaufgabe. Nehmen Sie das als möglichen Tipp mit.)