Karl Schlotmann Stiftung

SKRIPT ZUM

MIKROCONTROLLER PROJEKT DER KARL-SCHLOTMANN-STIFTUNG

INHALTSVERZEICHNIS

1. MOTIVATION

Schüler von weiterführenden Schulen sollen für Technik begeistert werden. Wir glauben, dass viele Schüler sehr geübt sind im Umgang mit Computern und einige Programmierkenntnisse haben. Diese sind jedoch nicht zwingend notwendig. Durch das Mikrocontroller-Projekt soll die Brücke geschlagen werden zwischen Programmierung auf der einen Seite und der Hardwareimplementierung auf der anderen Seite. Somit soll das Zusammenspiel von Soft- und Hardware verstanden werden. Die Arduinoplattform bietet eine gute Basis dafür, da sie günstig ist, mit ihrer einfachen Sprache sehr zielorientiert ist und über eine große Community verfügt, sodass viele Projekte bereits vorhanden sind.

2. DER ARDUINO

Der Arduino ist ein winziges Computersystem bestehend aus einer Entwicklungsumgebung, in der wir später die Programme schreiben, und der Hardware, dem Arduinoboard. Das Board kombiniert einen Mikrocontroller mit einem Takt von 16MHz mit vielen Schnittstellen zur Ansteuerung und Wechselwirkung mit Sensoren und Aktoren. Mit dem Board können wir zum Beispiel Knöpfe, LEDs, Helligkeitssensoren, GPS-Sensoren, WIFI-Schnittstellen, LCD-Displays und Motoren ansteuern und auslesen. Softwareseitig sind sehr viele Hilfen für die Ansteuerung vorhanden die uns die Arbeit erleichtern. Der Mikrocontroller entspricht in etwa einer CPU, worin die Befehle verarbeitet werden, und das Board einem Mainboard in einem PC.

2.1 DAS BOARD

Auf dem Board befindet sich eine DC-Buchse für die Spannungsversorgung und eine USB-Schnittstelle. Das Board kann meist über das USB-Kabel mit Energie versorgt werden, bei leistungshungrigen Anwendungen wie zum Beispiel Motoren ist eine eigene Energieversorgung notwendig. Über die Power-Anschlüsse kann die Spannung weiter an andere Schaltungen gegeben werden. Neben dem USB Anschluss befindet sich ein weiterer Mikrocontroller, der Programmer, der die Befehle der USB-Schnittstelle für den eigentlichen Microcontroller übersetzt.

Das Board hat 16 Analog Anschlüsse, die als Ein- oder Ausgang verwendet werden können. Diese Anschlüsse können Spannungen von 0V bis 5V ausgeben mit einer Genauigkeit von 10 Bit. (Damit entspricht also 0 => 0V und 10^{10}-1 =1023 => 5V). Im Gegensatz dazu können die 54 Digitalen Anschlüsse nur zwei Pegel ausgeben, entweder HIGH => 5V oder LOW => 0V. Einige dieser Anschlüsse beherrschen Puls-Weiten-Modulation, aber dazu später. Außerdem befindet sich auf dem Board noch eine grüne Power LED, die uns die korrekte Spannungsversorgung anzeigt, und eine eingebaute LED, die mit dem digitalen Pin 13 verbunden ist und für Testzwecke verwendet werden kann.

Falls ein Programm mal abstürzt, kann mit dem roten Reset-Knopf das Board neugestartet werden.

2.2 DIE ARDUINO IDE

In der Entwicklungsumgebung oder IDE (integrated development environment) werden wir unsere Programme schreiben. Zuerst prüfen wir dazu, ob der richtige Mikrocontroller (Werkzeuge->Platine->Arduino Mega oder Mega2560) und ob der richtige Port ausgewählt ist (Werkzeuge->Port, zB /dev/cu.usbmodem1411 (Arduino Mega oder Mega2560)). In der Statusleiste finden wir die Knöpfe:

- Verifizieren: Prüft die Syntax des Programms. Dazu muss das Board nicht angeschlossen sein.
- Hochladen: Lädt das Programm auf das Board hoch. Sobald es hochgeladen ist, wird der Arduino versuchen das Programm auszuführen, auch ohne an den PC angeschlossen zu sein.
- Neu: erstellt einen neuen Programmentwurf (Sketch).
- Öffnen: öffnet einen neuen Sketch. Es befinden sich bereits viele Standardprogramme inklusive.
- Speichern: speichert den eigenen Sketch.
- Serieller Monitor: öffnet ein Terminalfenster, in dem wir Daten während des Ausführens eines Programms
  an den Arduino senden können und empfangen können. Dies ist insbesondere sehr nützlich, um die Werte von Variablen auszugeben und somit das Programm zu debuggen.

In der Syntax muss jeder Befehl mit einem Semikolon „;“ beendet werden. Funktionen, Abfragen und Schleifen werden mit geschweiften Klammern angefangen und beendet { }. Übergebene Werte oder Bedingungen stehen in runden Klammern ().

Zur Erklärung der Programmstruktur siehe 3.1.

3. PROJEKTE

3.1 LED MIT DEM SERIAL MONITOR ANSTEUERN

Unser erstes einfaches Programm gliedert sich in drei Teile:

- Initialisierung von Variablen: Hier weisen wir bestimmten Datentypen einen Namen und ggf. einen Startwert zu. Der benötigte Speicherplatz wird dadurch reserviert. In unseren Projekten werden wir zunächst mit Integer und Char auskommen. Integer sind ganzzahlige Zahlenwerte zwischen -32,768(-2^{15}) bis 32,767 (2^{15}-1). Char speichert einen Buchstaben als Zahlenwert mit 8 Bit.
  Achtung: Nur Variablen, die außerhalb von Funktionen initialisiert wurden, können von allen Funktionen verwendet werden.
- Setup: Dieser Programmteil wird nur einmal ausgeführt. Dort teilt man dem Board mit, welche Anschlüsse als Ein- und welche als Ausgänge benutzt werden sollen und welche anderen Schnittstellen konfiguriert werden
- Loop: Die Funktion wird dauerhaft wiederholt und nach dem Durchlaufen direkt wieder neu gestartet.

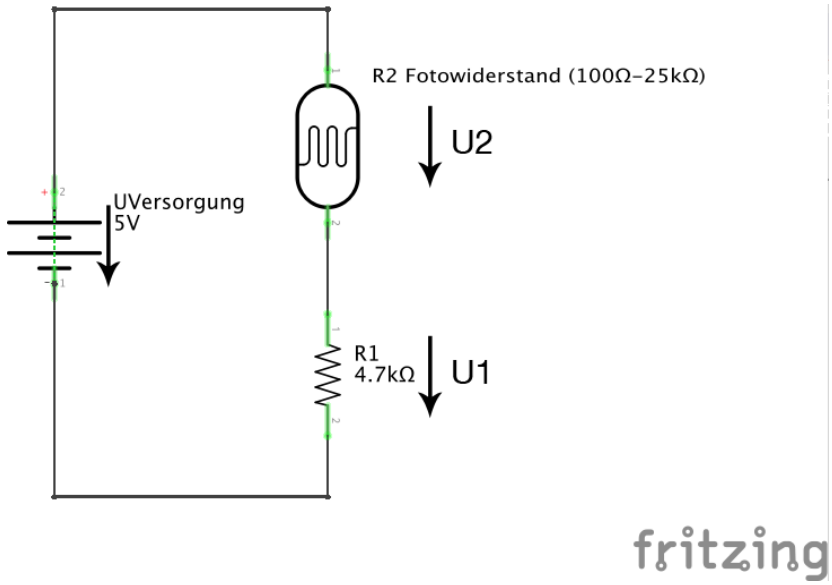
Fangen wir also an! Das Board soll die Status LED leuchten lassen, falls es über die serielle Schnittstelle ein „H“ empfängt, die LED ausschalten, falls ein „L“ empfangen wird und falls ein anderes Zeichen oder keines empfangen wird, die LED blinken lassen.

Dafür stehen euch folgende Befehle und Funktionen zur Verfügung:

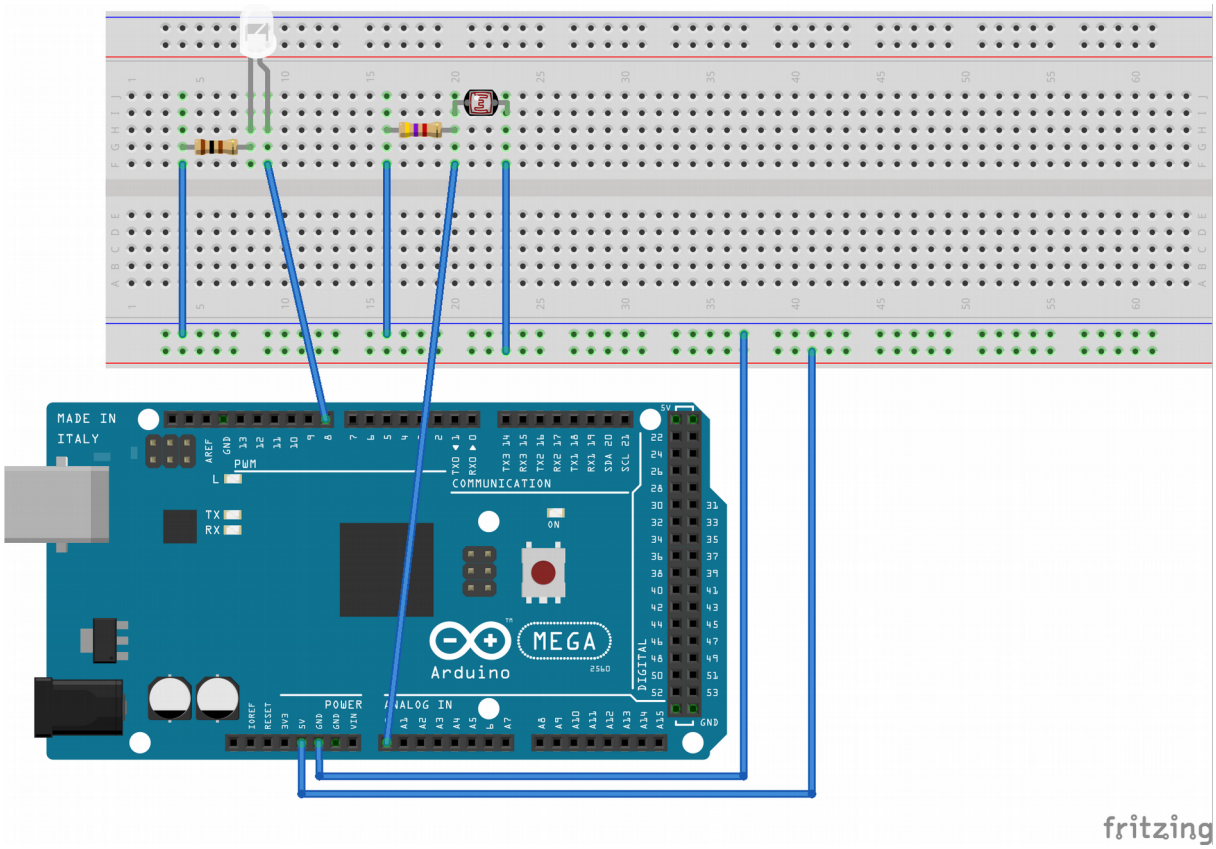
- Serial.begin(9600) : startet den Serial Monitor mit einer bestimmten Symbolrate. Der übergebene Wert muss gleich der angegeben Symbolrate am PC sein (im Fenster Serieller Monitor unten rechts zu finden).
  Wir werden hier immer 9600 verwenden.
- pinMode(PIN, ART) : legt die digitalen Anschlüsse als Eingang oder Ausgang fest. PIN wird mit der Kennung des Anschlusses ersetzt, in diesem Fall zum Ansteuern der LED mit 13. ART legt die Art des Anschlusses fest, und wird mit INPUT oder OUTPUT ersetzt.
- char C='H' : initialisiert eine neue Char Variable mit dem Namen C und dem Zeichen H als Startwert. Es muss allerdings kein Startwert vergeben werden!
- Serial.read(): Liest ein Zeichen ein. Das Zeichen muss aber noch in einer Variable gespeichert werden.
- Serial.available(): Gibt die Anzahl der zulesenden Bytes des Serial Monitors zurück. In unserem Fall reicht es, zu prüfen, dass dieser Wert größer als 0 ist.
- digitalWrite(PIN, PEGEL): Zieht den PIN auf HIGH (5V) oder auf LOW (0V). PEGEL wird also mit HIGH oder LOW ersetzt.
- delay(ZEIT): pausiert das Programm für die angegebene Zeit in Millisekunden.
- Serial.print("TEXT"): Gibt einen Text im Serial Monitor aus. Mit „\n“ wird eine neue Zeile angefangen. Alternativ kann man auch Serial.println("TEXT") benutzen, dort wird automatisch ein Zeilenumbruch eingefügt.
- if...else:
  ```
  if (BEDINGUNG){
    BEFEHL;
  }
  else if (ANDERE BEDINGUNG){
    ANDERER_BEFEHL;
  }
  else {
    NOCH_EIN_ANDERER_BEFEHL;
  }
  ```
  BEDINGUNG wird durch Vergleichsoperatoren ersetzt. Falls diese Bedingung zutrifft, wird der Befehl in den geschweiften Klammern ausgeführt. Mit else if kann eine andere Bedingungen geprüft werden. Falls keine der beiden erfüllt ist, wird der Befehl nach else ausgeführt. Die else Abfragen sind beide optional.
- Kommentare: Text der nicht vom Compiler als Programmcode interpretiert wird, können mit //Kommentar (bis zum Zeilenende) oder */Kommentar/* (über mehrere Zeilen) in das Programm eingefügt werden
- Vergleichsoperatoren: werden in Verbindung mit Abfragen oder Schleifen benutzt
  - == : Gleichheit
  - != : Ungleichheit
  - < : kleiner
  - > : größer
  - <= : kleiner gleich
  - >= : größer gleich

3.2 FOTOWIDERSTAND AUSLESEN UND DAMIT LEDS DIMMEN

Im folgenden Teilprojekt wollen wir die Helligkeit im Raum für den Mikrocontroller messbar machen und je nach Helligkeit LEDs heller oder dunkler leuchten lassen.



Dafür benutzen wir das Prinzip des Spannungsteilers. Wenn zwei Verbraucher (in unserem Fall Widerstände) in eine Reihenschaltung angeschlossen werden, muss durch beide Verbraucher der gleiche Strom fließen. Durch das Ohmsche Gesetz ergibt sich: $I=U_{VERSORGUNG}/R_{GES}$ (1) . R_{GES} setzt sich aus der Addition von beiden Widerständen zusammen. $R_{GES}=R_1+R_2$ (2). Gleichzeitig müssen die Teilspannungen über die Widerstände die Versorgungsspannung ergeben $U_{VERSORGUNG}=U_1+U_2$ und der Strom ist auch hier über das Ohmsche Gesetz bestimmbar. $I=U_1/R_1=U_2/R_2$ (3). (3) in (1) ergibt nach Umstellen: $U_1=U_{VERSORGUNG}* R_1/R_1+R_2$. Diesen Zusammenhang können wir nutzen, da der Widerstand unseres Helligkeitssensors abhängig ist von der Helligkeit. Wir müssen also nur die Spannung zwischen beiden Widerständen auslesen und können dadurch die Helligkeit bestimmen. $U_{VERSORGUNG}$ ist 5V, je nach Helligkeit kann also U_1 beliebige Werte zwischen 0V und 5V betragen, zum Beispiel 2.9V. Der Fotowiderstand R1 und R2 sollten in etwa gleich groß sein, wir werden deswegen für R2 einen 4,7kΩ Widerstand benutzen. Da es viele unterschiedliche Werte für U_1 und nicht nur zwei Zustände gibt, müssen wir einen der analogen Eingänge des Arduino benutzen. Die LEDs werden an die digitalen Eingänge angeschlossen. Da diese nur 5V oder 0V ausgeben können, die LEDs aber eine Spannung zwischen 2-4V benötigen, schalten wir einen Widerstand mit den LEDs in Reihe. Durch die LEDs kann der Strom nur in eine Richtung fließen, deswegen wird die + Seite mit dem längeren Beinchen zum digitalen Pin angeschlossen und die - Seite Richtung Ground. Die Widerstände für die LEDs wählen wir wie folgt: Weiß - 100Ohm, Rot – 200 Ohm,Gelb - 200Ohm,Grün - 100Ohm und Blau - 100Ohm.



Wir wollen die Helligkeit der LED über einen der digitalen Eingänge regeln. Diese können jedoch nur HIGH und LOW als diskrete Pegel ausgeben. Uns steht jedoch Puls-Weiten-Modulation zur Verfügung. Dabei wechselt der Eingang sehr schnell im Millisekundenbereich die Pegel. Da das menschliche Auge nur eine begrenzte Anzahl von Bildern pro Sekunde wahrnimmt, nehmen wir die LED trotzdem als durchgehend leuchtend wahr. Falls der Pegel die meiste Zeit an ist, wird uns die LED heller erscheinen, als wenn der Pegel die meiste Zeit aus ist.

Zusammengefasst soll unser Programm also folgendes tun:

Die Spannung wird am Fotowiderstand messen, der Wert wird abspeichert und am Besten über den Serial-Monitor ausgegeben. Danach wird der verrechnete Wert an den digitalen Eingang zur Puls-Weiten-Modulation übergeben. Tipp: lasst euch den Sensorwert ausgeben und bastelt eine Rechenvorschrift, so dass höchste Sensorwert auf 255 abgebildet wird und der niedrigeste Sensorwert auf 0. Zudem solltet ihr danach eine Abfrage schreiben, um PWM-Werte kleiner als 0 oder größer als 255 abzufangen. Falls euch unklar ist, wo welche Programmteile stehen müssen, schaut doch nochmal bei Projekt 3.1 nach.

Das sind die Befehle die ihr zusätzlich für das Projekt braucht:

- analogRead(EINGANG): Liest einen Wert über den angegeben Analogeingang ein, in unserem Fall A0. Der zurückgegebene Wert muss in einer Integer Variable gespeichert werden und liegt zwischen 0 und 1023.
- analogWrite(LED,WERT): Diese Funktion kümmert sich um die Puls-Weiten-Modulation an unserem digitalen Ausgang. Der übergebene Wert muss zwischen 0 und 255 sein.

3.3 LCD-DISPLAY

In diesem Teilprojekt wollen wir ein LCD-Display mit 16x2 Zeichen ansteuern. Eigentlich müssten wir dafür fast jeden der Anschlüsse über unserem LCD mit einem eigenen Pin an unserem Arduino verkabeln. Um uns diesere Arbeit zu ersparen und mehr Pins zur freien Verfügung zu haben, verwenden wir ein I^2C-Backpack. Dies ist eine Schnittstelle, die die Befehle für die verschiedenen Pins am LCD übersetzt von zwei analogen Pins übersetzt. Gleichzeitig hat jedes I^2C-Gerät eine eigene Adresse, so dass wir acht LCD-Displays parallel betreiben könnten. Der SDA und der SCL Pin befinden sich auf dem Arduino auf den digitalen Pins 20 und 21.

Folgende Befehle brauchen wir für diesen Versuch:

- LiquidCrystal_I2C lcd(0x20,16,2): Setzt die I^2C- Adresse auf 0x20 und gibt an, dass es ein 16x2 Zeichen Display angeschlossen ist. Achtung: Dieser Befehl muss außerhalb von setup() stehen, damit auch andere Funktionen auf die erstellte Variable lcd zugreifen können.
- #include <LIBRARY.h>: Dieser Befehl fügt eine Bibliothek in unser Programm ein, dass uns viele Funktionen bereitstellt und uns so viel Arbeit abnimmt. Hier brauchen wir als LIBRARY Wire.h und LiquidCrystal_I2C.h. Achtung: nach #include Befehlen folgt kein Semikolon!
- lcd.init(): initialisiert das Display
- lcd.backlight(): schaltet die Hintergrundbeleuchtung an
- lcd.print("TEXT"): gibt einen Text aus. Es kann auch ein Char oder ein Integer übergeben werden.
- lcd.setCursor(0, 1): Setzt den Cursor auf ein Zeichen, ab dort wird neuer Text eingefügt. Der erste Wert ist die Spalte von 0 bis 15 und der zweite Wert ist die Zeile, also 0 oder 1.
- lcd.clear(): löscht den Inhalt des Displays und setzt den Cursor auf Position 0
- lcd.home(): setzt den Cursor auf Position 0

3.4 FUNKTIONEN

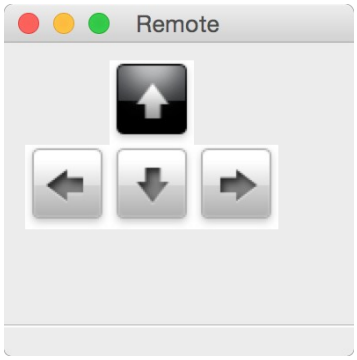
Die Aktionen aus den letzten Teilprojekten wollen wir später auch bei unserem RC-Auto benutzen. Damit unser Programmcode schön strukturiert bleibt und wir wenige Wiederholungen haben, wollen wir nun diese Aktionen in Funktionen einbetten. Die setup() und die loop() Funktionen sind bereits solche. Eine Funktion definieren wir beispielsweise wie folgt:

```
int NAME(VARIABLE_INT_1, VARIABLE_INT_2){

  int RÜCKGABE;
  BEFEHL;
  return RÜCKGABE;

}
```

Der Aufruf der Funktion würde über NAME(VARIABLE_EXT_1,VARIABLE_EXT_2); erfolgen. Falls die Funktion keine Rückgabe liefern soll, ist der Typ der Funktion void und ihr könnt keinen return Befehl schreiben. Die Variablen die in der Funktion definiert werden, sind nur für diese sichtbar. Ebenso speichert die Funktion nur eine Kopie der Variablen und speichert Änderungen in diese Kopien, nicht in die originalen Variablen.

Wir wollen also nun unsere bisherigen Projekte in Funktionen auslagern, die über loop() aufgerufen werden. Überlegt eucht dazu, welche Variablen übergeben werden müssen. Die Befehlsstruktur der Teilprojekte könnt ihr dafür beibehalten. Allerdings sollte das Einlesen eines Zeichens aus dem Serialmonitor in der loop()-Funktion bleiben.

3.5 EIN RC-CAR STEUERN



Endlich! In unserem nächsten Projekt wollen wir ein RC-Auto zum Fahren bringen. Dafür brauchen wir drei neue Teile: ein XBee-Dongle als Sender für den PC, ein XBee-Shield als Empfänger und einen Motortreiber. Für den Rechner habt ihr ein eigenes Programm als Steuerung, dass die Tastaturbefehle erfasst und entsprechende Zeichen wie zum Beispiel „w“ über Funk überträgt. Ihr könnt auch über das Klicken der Buttons steuern. Das Programm ist so ausgelegt, dass die Buttons beziehungsweise die Tasten nicht dauerhaft gedrückt werden. Auf dem Arduino empfängt das XBee-Shield die Zeichen und leitet sie über die Serial-Schnittstelle an den Mikro-controller weiter. Für den Mikrocontroller macht es also keinen Unterschied, ob die Zeichen über USB oder über das XBee-Shield übertragen werden. Ein weiteres Bauteil ist der Motortreiber. Dieser kommuniziert ebenso wie unser LCD-Display über I^2C.

- Um die Geschwindigkeitswerte zu speichern, werden wir einen Array benutzen. Ein Array ist eine Tabelle von Werten eines gleichen Datentyps. Die Initialisierung für den Dateityp Integer geschieht über: int NAME[4] = {WERT1, WERT2, WERT3, WERT4}; Der Array NAME hat nun vier Startwerte. Auf diese Werte kann man über NAME[i] zugreifen. i gibt das Feld an, jedoch beginnt die Zählung bei 0. In diesem Fall würde man also mit NAME[0] auf den WERT1 und mit NAME[3] auf WERT4 zugreifen.
- Um die Zuweisung der Geschwindigkeitswerte zusammenzufassen, benutzen wir for-Schleifen als Kontrollstruktur mit einer bestimmten Wiederholungsanzahl:
  ```
  for (int i = 0; i < 4; i++) {
  m[i]=WERT;
  }
  ```
  In den runden Klammern geschieht folgendes: zuerst wird eine Schleifenvariable mit Startwert initialisert, dahinter steht die Bedingung, die vor jedem Durchlauf geprüft wird und dahinter die Anweisung für die Schleifenvariable. Die Schleifenvariable können wir auch in der Schleife benutzen, hier zum Beispiel um auf die Werte unseres Arrays zuzugreifen.
- Für die Statusvariablen benutzen wir eine Schaltvariable, zum Beispiel bool NAME= true;
  Die Variable kann nur die Werte true oder false annehmen und lässt sich mit NAME=!NAME; umkehren. Außerdem kann man sie leicht in if-Abfragen einbauen, beispielsweise mit if(NAME) oder if (!NAME). Beim ersten Beispiel wird die Abfrage ausgeführt falls, NAME = true ist, beim zweiten Beispiel, falls NAME=false ist.

- Für die Motoren brauchen wir folgende Befehle:
  - Motor1->run(BACKWARD) oder Motor1->run(FORWARD); Dieser Befehl gibt die Laufrichtung der Motoren an.
  - Motor1->setSpeed(M[0]); übergibt den Wert von M[0] als Geschwindigkeit

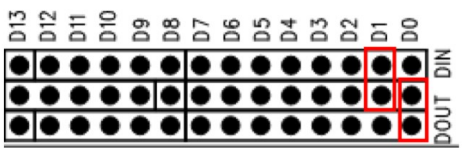
Wir haben bereits einen Sketch für euch vorbereitet. Die eigentliche Motorsteuerung soll in zwei Schritten erfolgen. void MotorWert(char H) bekommt ein Zeichen übergeben und berechnet mit diesem die neuen Geschwindigkeitswerte.

Diese soll folgendes machen: Falls das Auto ein „w“ erhält und die Richtung bereits vorwärts ist, soll die Geschwindigkeit erhöht werden. Achtung: die Maximalwerte sind 250 und 0! Falls die aktuelle Richtung rückwärts ist, soll die Geschwindigkeit stark gedrosselt werden. Für „s“ funktioniert es genau andersherum. Falls ein „a“ empfangen wird, sollen die Motoren auf der linken Seite des Autos schrittweise gedrosselt werden, aber nicht komplett stillstehen. Für „d“ geschieht das gleiche auf der rechten Seite. Falls in keine Richtung gelenkt wird, sendet der PC ein „f“. Die Funktion soll dann, solange eine Seite der Motoren langsamer dreht, deren Geschwindigkeit solange erhöhen, bis alle Motoren wieder gleichschnell laufen. Falls keine Taste gedrückt ist, wird ein „r“ übertragen und das Auto soll ausrollen. Dafür wird die Geschwindigkeit aller Motoren gedrosselt, bis diese stillstehen. Sobald alle Motoren stillstehen wird die Variable D für die Richtung umgekehrt. Damit beim Stillstand nicht ständig die Richtung umgekehrt wird, benutzen wir eine weitere Statusvariable Z. Diese gibt an, ob der letzte Aufruf die Richtungsumkehrung war und wird deaktiviert, sobald ein Steuerungsbefehl empfangen wird. Falls das Fahrzeug in eine Richtung gelenkt wird, sollen die Blinker aktiviert werden. Ebenso soll das Bremslicht beim Bremsen aktiviert werden. Ihr könnt dafür die LEDs parallel verschalten. void fahr() übergibt dann je nach Richtung, also Status der Variable D, die Werte an den Motortreiber.

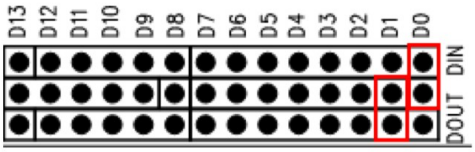
Um den Arduino mit Spannung mitzuversorgen, muss der Vin Jumper auf dem Motorshield gesetzt werden.

Beim aufgesteckten XBee-Shield müssen die Jumper wie folgt gesetzt werden:

Zum Fahren des Autos:



Zum Hochladen eines Sketches per USB:



4. KONFIGURATION

4.1 XBEE

- Ein XBee Radio-Modul in USB-Explorer einsetzen und XCTU öffnen, neues Radio Modul hinzufügen
- XBEE Modul konfigurieren: Sender und Empfänger müssen beide die gleiche PAN-ID haben. Am Besten markiert man sich mit Filzstift die jeweils zueinander gehörenden Sender und Empfänger.

Die Destination Adress High ist 0013A200 und die Destiantion Adress Low ist mit der untersten fettgedruckten Zeile auf dem jeweilig anderen Radio Modul zu wählen, z.B. 40AFBBC9.
Eins der Module muss später als Coordinator am USB-Explorer betrieben werden. Dafür wird in XCTU die Firmware auf die Coordinator-Firmware geflasht.

- Falls es zu Störungen beim Betrieb von mehreren RC-Autos kommt, kann die PAN-ID paarweise geändert werden.
- Vorsicht beim Wechseln der Radio-Module: Diese vorsichtig, mit leichtem Wackeln parallel zu den Stiftleisten, rauslösen. Die Pins verbiegen sehr leicht!

4.2 PROGRAMME

- Python: entweder mitgeliefertes Programm ausführen oder Daten aus der Linksammlung bezüglich Python herunterladen und den Quellcode ausführen
  Installation der Erweiterungen unter Mac im Terminal: „cd“ auf das Verzeichnis des entpackten Ordners, danach Installation mit „python setup.py install“. Ein Account mit Administratorrechten wird benötigt.
- Arduino: Libraries hinzufügen (Sketch->Include Library->Add .ZIP Library)

5. HÄUFIGE FEHLER

- Der Serialmonitor gibt nur komische Zeichen aus.
  Lösung: gleiche Baudrate im Fenster sowie im setup() wählen. Wir nehmen werden immer 9600 wählen.
- 'i' was not declared in this scope
  Lösung: eine Variable die verwendet wird, wurde nicht im Programm definiert
- Syntaxfehler: _3.5_Motoransteuerung.ino: In function 'void loop()':
  _3.5_Motoransteuerung.ino:256:3: error: expected ';' before 'sensorWert'
  Lösung: „In function“ gibt die Funktion an, in der der Fehler auftritt. In der nächsten Zeile steht, welcher Fehler auftritt und in welcher Zeile, in diesem Fall 256. Die Zeile des Programmcodes wird euch unten links in der Arduino IDE angezeigt.
- avrdude: ser_open(): can't open device "/dev/cu.usbmodem1421": No such file or directory
  oder avrdude:time_out()
  Lösung: entweder habt ihr den Arduino nicht angeschlossen, auf dem XBee-Shield sind die Jumper falsch gesetzt (siehe dazu Bild im Abschnitt 3.5) oder der Arduino hat sich aufgehangen. Im letzten Fall hilft oft die Reset-Taste oder die Motoren von der Spannungsversorgung zu trennen und den Arduino per USB aus und wieder einzustecken.
- Ein Motor dreht falsch herum.
  Lösung: Motori->run(BACKWARD); durch Motori->run(FORWARD); ersetzen oder + und – des Motors vertauschen

6. LINKSAMMLUNG

- Download der Arduino IDE: http://www.arduino.cc/en/Main/Software

- Arduino Reference: http://www.arduino.cc/en/Reference/HomePage
- 4WD-Plattform: https://www.dfrobot.com/wiki/index.php?title=NEW_A4WD_Mobile_Robot_with_encoder_ %28SKU:ROB0025%29
- XBee-Konfiguration: http://examples.digi.com/get-started/basic-xbee-zb-zigbee-chat/ http://examples.digi.com/get-started/configuring-xbee-radios-with-x-ctu/
- Python:
  - Python 2.7: https://www.python.org/downloads/
  - wxPython: http://www.wxpython.org/
  - PySerial: http://pyserial.sourceforge.net/
  - PyCharm: http://www.jetbrains.com/pycharm/
- Datenblätter, Dokumentationen und Libraries:
  - Arduino Mega2560: http://www.arduino.cc/en/Main/ArduinoBoardMega2560
  - I^2C-LCD-Display: http://www.dfrobot.com/wiki/index.php/I2C/TWI_LCD1602_Module_%28Gadgeteer_Compatible%29_%28SKU:_DFR0063%29
  - Motorshield: https://learn.adafruit.com/adafruit-motor-shield-v2-for-arduino/overview
  - XBee-Shield:

    ftp://imall.iteadstudio.com/IM120417004_XBeeShield/DS_IM120417004_XBeeShield.pdf

7. PROJEKTVERANTWORTLICHE

- Alexander Stramma alexander.stramma@karl-schlotmann-stiftung.de (Elektronik, Skript und Projekte)
- Weitere Kontaktmöglichkeit: projekte@karl-schlotmann-stiftung.de