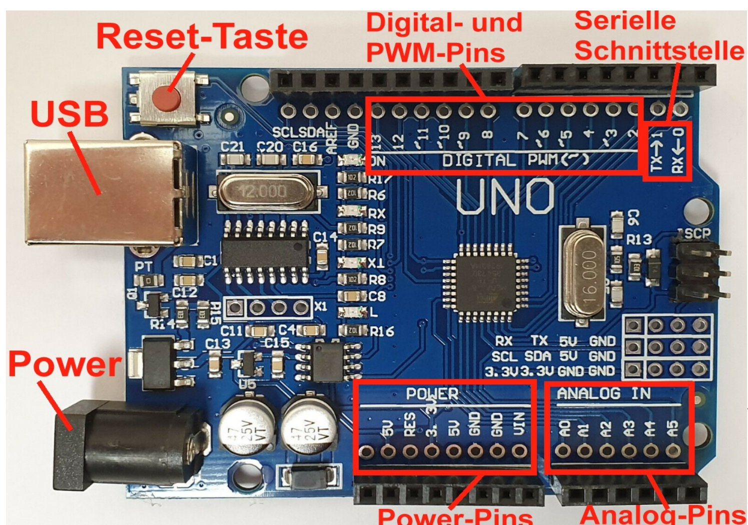
Einstieg in die Arduino-Programmierung für Amateurfunker
von Gunthard Kraus, DG8GB

(Version V1.0 vom 26. Januar 2020)

1. Worum geht es?
Microcontroller sind längst in die meisten Bereiche des Alltags eingedrungen – warum sollte es beim Amateurfunk anders sein? (Das zeigen auch die Artikel in den UKW-Berichten, z. B. von Wolfgang Schneider, DJ 8 ES).
Nur haben wir es hier bei den Anwendern mit Leuten zu tun, die technisch interessiert sind, Projekte aushecken oder tüfteln, aber auch die Programmierung dieser Teilchen gerne selbst vornehmen möchten. Ohne fundierte Grundkenntnisse geht das leider nicht, die nötige C-Programmierung wirkt oft abschreckend und deshalb sollen hier anhand des kleinen Projektes
Programmierung eines „cq-Rufes in CW mit einem Arduino Uno“
die Berührungsängste genommen, die Einstiegshürden überwunden, die nötigen Grundlagen vermittelt und Mut + Freude für weiteres selbständiges Arbeiten geweckt werden.

2. Arduino – was ist das?
Dahinter versteckt sich ein Konzept von drei verschieden starken Brüdern einer italienischen Familie von sofort betriebsfertigen Microcontroller-Platinen (**Uno / Leonardo / Mega 2560)**, die in der Zwischenzeit längst geklont wurden und deshalb preislich ab etwa 5 Euro beim „Uno“ beginnen. Es gibt hierzu inzwischen im Internet unzählige Anwendungen, Programmbeispiele und Projektbausätze (Fachausdruck: „Shields“), jede Menge Communities und zwischen 1000 und 2000 Angebote beim Aufruf von „arduino uno“ in Ebay.



Wir starten also in diesen Kurs mit einem für deutlich weniger als 10 Euro bei Ebay beschafften Einsteiger-Modell „Uno“ und bekommen das zu sehen.
Es handelt sich um einen kompletten kleinen Rechner mit **USB-Verbindung** zum PC, einer Reihe von **digitalen Ein-/ Ausgängen** sowie **Pins für Pulsweiten.Modulation.**
Ebenso gibt es **analoge Ein-/ Ausgänge sowie eine Serielle Schnittstelle.**
Gespeist wird die Maschine entweder von den in der USB-Verbindung bereitgestellten +5V oder über ein externes Steckernetzteil und den Power-Eingang. Ein Festspannungsregler auf dem Board sorgt dann für die richtige System-Betriebsspannung von +3,3 V.
Wie jeder anständige Computer besitzt der Uno einen Programmspeicher und einen Arbeitsspeicher – und noch einige andere Spielereien wie einen 10 Bit-AD-Wandler usw. Doch das soll uns vorerst nicht weiter interessieren, denn wir brauchen für unsere Arbeit zuerst eine so genannte **„Entwicklungsumgebung“**, also eine **IDE = „integrated developement enviroment“ (Eingabe: „arduino software“)** aus dem Internet.

3. Grundsätzlicher Aufbau von „C“ und Arduino-Programmen

Da gibt es feste Spielregeln und verbindliche Vereinbarungen. Die muss man kennen, sich aneignen, immer bereit halten und pingelig anwenden. Dann kann man nach diesem Kurs auch in wesentlich kompliziertere Programmbeispiele einsteigen und die nächste Stufe (leider meist mit weiteren neuen Informationen und Vorschriften) erklimmen. Anders geht es nicht....

Prinzipielle Programmstruktur:

A) Kommentare

```
/*      so müssen längere Erlauterungen und Kommentare eingefügt werden. Das
        Programm ignoriert sie dann
*/
```

```
//      so wird eine einzige Kommentarzeile innerhalb eines Programm eingeschoben
```

==

b) Deklarationen

Das können **Konstanten** (= Zahlenwerte, meist **„int“ = „integer“ = ganzzahlig**) sein. Mit ihrer Hilfe weist man z. B. einem Portpin einen **Namen** zu, um nicht immer mit der Portpin-Nummer arbeiten zu müssen.

Beispiele für solche Zeilen:

```
const  int   ledPin = 13;          // an Portpin 13 (ledpin) hängt eine interne LED

const   int   buttonPin = 2;       // an Portpin 2 (buttonpin) ist ein Taster angeschlossen
```

Ebenso können **Variablen** (z. B. Messergebnisse) durch einen leicht verständlichen Namen angesprochen werden.

Beispiel:

```
int buttonState = 0;               // Variable zum Speichern des Tasten-Status
```

==

c) Setup-Routinen

Damit **initialisieren wir** (= wir weisen bestimmten Pins bestimmte Aufgaben zu).
Beispiel:

```
void setup()          {
  pinMode(ledPin, OUTPUT);              // ledPin wird als Ausgang betrieben
  pinMode(buttonPin, INPUT);            // buttonPin ist nun ein Eingang
}
```

Es gilt:

Routinen bzw. Funktonen beginnen immer mit „**void**“ und dann folgt erst der **Name**, an derm **zwei runde Klammern** hängen. Zwischen diesen beiden Klammern können wir **Werte oder Variablen übergeben**, die innerhalb der Routine benötigt werden.
Zwischen **zwei geschweiften Klammern** stehen dann die einzelnen Programmschritte (Fachausdruck: Anweisungen).

Vorsicht: Routinen- und Funktionen-Namen dürfen NIE mit einer Zahl beginnen!

==

d) Prototypen der verwendeten Funktionen
Man sollte das Hauptprogramm immer so einfach wie möglich halten (….das erleichtert die Fehlersuche....) und soviel Funktionen wie möglich in eigene Unterprogramme packen. Das sind z. B. Zeitverzögerungen, Pin-Abfragen, Sendungen über die serielle Schnittstelle, Messungen eines Analogwertes, Text-Erzeugung usw. Sie können nun vom Hauptprogramm aufgerufen werden.

ALLE diese Unterprogramme müssen als Prototypen VOR dem Hauptprogramm angeordnet (= angemeldet) sein, damit sie überhaupt erkannt werden können.

Wie solche Routinen geschrieben sein müssen, haben wir eben gelernt (siehe obiger Rahmen). **Sehen wir uns als Beispiel mal die Routine „Ausgabe des Morsezeichens „e = dit“ als Puls mit 100 Millisekunden Länge, gefolgt von einer Pause mit 100 Millisekunden“ in Form einer Endlos-Schleife an.**

Da ist es gut, dass es für die **Zeitverzögerungen bereits eine interne fertige Funktion „delay“** gibt, der wir in der runden Klammer nur die **Verzögerung in Millisekunden** übergeben müssen.

Den gewünschten Pegel an einem digitalen Portpin geben wir mit der Anweisung **„digitalWrite“** aus. In runden Klammern folgt zuerst die Pin-Nummer oder der zugeordnete Namen, gefolgt vom Pegel. So sieht der Prototyp dann aus, da der LED auf dem Board schon intern der Name **„LED_BUILTIN“** zugeteilt ist.

==

```
void dit()     {
  digitalWrite(LED_BUILTIN, HIGH);          // LED einschalten
  delay(100);                               // 100 Millisekunden leuchten lassen
  digitalWrite(LED_BUILTIN, LOW);           // LED ausschalten
  delay(100);                               // nochmals 100 Millisekunden warten
}
```

==

Ein guter Rat:
Die Sache mit dem **Kommentar am Ende jeder Programmzeile** sollte man sich unbedingt angewöhnen. Man stößt nämlich eine ganze Serie unfeiner Formulierungen aus, wenn man ein aufwendiges Projekt nach einem Jahr wieder ausgräbt und absolut nicht mehr zusammenbringt, was man sich da bei manchen Zeilen gedacht hat....eigene Erfahrung....

e) Und jetzt folgt das komplette Programm zur Ausgabe von „dit“ in CW.

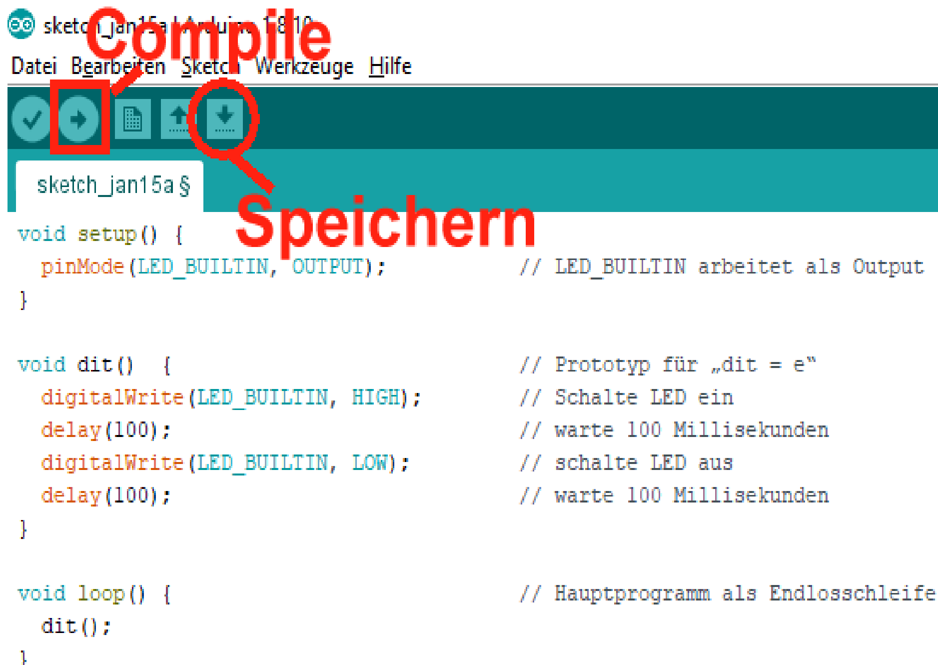
Allerdings müsen wir vorher noch den **Pin 13 = „LED_BUILTIN“** mit der eingebauten LED auf **OUTPUT** schalten!

```
==========================================================
void setup() {
  pinMode(LED_BUILTIN, OUTPUT);          // LED_BUILTIN arbeitet als Output
}


void dit()  {                             // Prototyp für „dit = e“
  digitalWrite(LED_BUILTIN, HIGH);        // Schalte LED ein
  delay(100);                             // warte 100 Millisekunden
  digitalWrite(LED_BUILTIN, LOW);         // schalte LED aus
  delay(100);                             // warte 100 Millisekunden
}


void loop() {                             // Hauptprogramm als Endlos-Schleife
  dit();
}
===========================================================
```

Jetzt wird es ernst und wir wollen diese „dit“-Folge auf unserem Uno zum Laufen bringen.

4. Test des fertigen Programms

Schritt1:
Arduino - IDE starten und im linken oberern Eck auf „**Datei“ und „Neu**“ klicken (...oder gleich auf die dritte grüne Taste von links, also rechts neben Compile).
Es öffnet sich ein bereits vorbereitetes Blatt („sketch“). **Das obige Programm können wir nun durch simples Kopieren aus dem Manuskript in dieses neue Blatt einfügen, nachdem wir dort alle alten Einträge gelöscht haben.**



Dann wird gespeichert und dabei ein Name vergeben. Der Rest ist pure Freude, denn das Programm wird nun kompiliert, automatisch zum Board übertragen und gestartet.
Jetzt sollte die LED auf dem Uno-Board als „Dauer-dit“ blinken.

5. Programm-Erweiterung: der Text „de“ als Endlosschleife

Ich habe mir dazu erst mal aus Wikipedia mit dem Begriff „Morsezeichen“ die genaue Vorgabe für eine korrekte Morse-Übertragung geholt:

Es gilt Folgendes:

- **Ein *Dah* ist dreimal so lang wie ein *Dit*.**
- **Die Pause zwischen zwei Symbolen eines Zeichens ist ein *Dit* lang.**
- **Zwischen Buchstaben in einem Wort wird eine Pause von der Länge eines *Dah* (oder drei *Dits*) eingeschoben.**
- **Die Länge der Pause zwischen Wörtern entspricht sieben *Dits*.**

Das ergibt folgende Vorgaben bei der Controller-Programmierung:

a) Ein „Dit“ ist 100 Millisekunden lang auf HIGH, dann folgen 100 Millisekunden LOW

b) Es muß „**Dah“ (mit 300 ms HIGH-Pegel und 100 ms LOW - Pegel) als Prototyp** neu geschrieben werden.

c) Es gilt ein Abstand von „**einem Dit“ = 100 ms LOW zwischen den einzelnen Symbolen** (Dit bzw. Dah) eines Morsezeichens.

d) Der Abstand zwischen den **einzelnen Zeichen eines Wortes ist „drei Dit auf LOW“. Also brauchen wir noch eine Pause „space_sign()“ von (300 ms - 100 ms)) 200 ms.**

e) Die Pause zwischen **zwei Worten soll „sieben Dit“ (= 700 ms auf LOW)** sein. Ein Dit hängt ja automatisch am Ende jedes Zeichen-Prototyps, deshalb schreiben wir noch einen **Prototyp „space_word()“ mit einer Länge von (700ms - 100 ms) = 600 ms LOW.**

So sieht das Programm dann aus:

```
void setup() {
  pinMode(LED_BUILTIN, OUTPUT);
}

void Dit()  {
  digitalWrite(LED_BUILTIN, HIGH);
  delay(100);
  digitalWrite(LED_BUILTIN, LOW);
  delay(100);
}

void Dah()  {
  digitalWrite(LED_BUILTIN, HIGH);
  delay(300);
  digitalWrite(LED_BUILTIN, LOW);
  delay(100);
}
```

```
void space_sign()  {
   delay(200);
}

void space_word()  {
  delay(600);
}

void d()  {
  Dah();
  Dit();
  Dit();
}

void e()  {
  Dit();
}


void loop() {
  d();
  space_sign();
  e();
  space_word();

  space_word();
}
```

Noch eine weitere persönliche Anmerkung und Emutigung des Autors:
Es ist gerade zu unglaublich, wieviel Schreibfehler Einem schon bei einem so einfachen Projekt unterlaufen! Selbst wenn das Programm im Prinzip in Ordnung ist, vergißt man gern einen Strichpunkt oder eine Klammer, setzt eine falsche Klammer, macht Rechtschreibfehler oder verwechselt Groß- und Kleinschreibung...usw...usw.
Da verweigert die IDE sofort das Compilieren (= Übersetzen in Maschinencode) und schreibt am unteren Bildrand rote Fehlermeldungen. Ausserdem wird die defekte Zeile mit dem Fehler rot markiert.
Ist alles OK, dann leuchtet die COMPILE-Taste gelb. Dann erst wird kompiliert, übertragen und automatisch gestartet.
Zum Trost: das passiert JEDEM (...und dem Autor fast jedes Mal). Aber wer das bestreitet, der lügt. Selbst langjährige Programmierer tun das.
Das zum Trost. Bitte einfach mit frohem Mut weitermachen, bis alles läuft.

6. Es geht mit „cq de“ weiter

Der Weg ist nun klar:

a) wir brauchen Prototypen für „c“ und „q“:

```
void c()  {
  Dah();
  Dit();
  Dah();
  Dit();
}

void q()  {
  Dah();
  Dah();
  Dit();
  Dah();
}
```

====================

b) Wir müssen das Hauptprogramm, also die Zeitschleife „void loop“, entsprechend erweitern:

===============

```
void loop() {
  c();
  space_sign();
  q();

  space_word();

  d();
  space_sign();
  e();

  space_word();
  space_word();

}
```

Hier haben wir das komplette lauffähige Programm. Bitte testen!

.

```
void setup() {
  pinMode(LED_BUILTIN, OUTPUT);
}

void Dit()  {
  digitalWrite(LED_BUILTIN, HIGH);
  delay(100);
  digitalWrite(LED_BUILTIN, LOW);
  delay(100);
}
```

```
void Dah()  {
  digitalWrite(LED_BUILTIN, HIGH);
  delay(300);
  digitalWrite(LED_BUILTIN, LOW);
  delay(100);
}

void space_sign()  {
   delay(200);
}

void space_word()  {
  delay(600);
}

void d()  {
  Dah();
  Dit();
  Dit();
}

void e()  {
  Dit();
}

void c()  {
  Dah();
  Dit();
  Dah();
  Dit();
}

void q()  {
  Dah();
  Dah();
  Dit();
  Dah();
}

void loop() {
  c();
  space_sign();
  q();
  space_word();

  d();
  space_sign();
  e();
  space_word();

  space_word();
}
```

7) Eine Aufgabe als Schlußtest für Sie

Sie lautet:
Ergänzen Sie nun das Programm soweit, dass der Text

cq cg cq de DG8GB   k“

als CW-Signal ausgegeben und an der LED angezeigt wird (...Sie dürfen auch gern Ihr eigenes Rufzeichen verwenden!)

Na, läuft es? Wenn ja, dann geben Sie sich selbst eine gute Note, bleiben dran und machen weiter. Wenn Nein: einfach die Fehler suchen und korrigieren (geht in der Arduino - IDE ganz leicht!).

8) Der Schritt zum fortgeschrittenen Programmierer

Wer noch nicht verzweifelt ist, sondern Lust auf mehr bekommen hat, der wage sich an die Erarbeitung von aufwendigeren Projekten. In der Arduino-IDE werden unter „Examples“ viele Beispiele mit steigendem Schwierigkeitsgrad oder ganz anderen Anwendungen mitgeliefert. Da kann man wühlen und sich einlesen – ganz abgesehen von den unzähligen „Shields“ (= Anwendungsprojekten) und Foren, die im Internet bereit stehen. Aber es gibt dabei halt noch viel zu lernen....
Für unser Beispiel würden sich folgende Varianten anbieten:

1) Schließen Sie an einen Digitalpin einen Drucktaster an. Wenn er kurz gedrückt wird, erfolgt einmal der „CQ-Ruf“.
--

2) Schließen Sie zwei Drucktaster an zwei getrennte Digitalpins an. Wird der erste nur kurz angetippt, dann wird der „CQ-Ruf“ pausenlos wiederholt. Ein Tipp auf den zweiten Taster stoppt die Aussendung und sendet einmal den Buchstaben „k“.
--

3) Erstellen Sie das komplette Alphabet samt Zahlen von 0 bis 9 als Prototypen und ändern Sie nun beliebig den ausgegebenen Text.

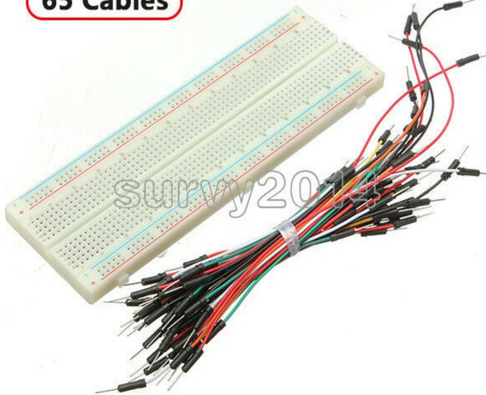
Achtung: Funktionen dürfen in „C“ NIE mit einer Zahl beginnen! Deshalb müssen die Prototypen (hier: für die Zahlen von 0 bis 9) IMMER mit einem Buchstaben anfangen.
Beispiel: „void 6()“ ist verboten und führt zu einer Fehlermeldung. Es muss also als

void six()

geschrieben werden.
---

4) Verwenden Sie weitere Drucktaster an Digitalpins, um 5 unterschiedliche Texte ausgeben zu können.

Noch ein Praxistipp:



Zum Austesten von Ideen und Programmen sollte man sich gleich ein solches **„breadboard“ (= Steckbrett) samt Kabelsatz** anschaffen (ca. 5 Euro). Darauf lassen sich mit bedrahteten Bauelementen schnell komplette Schaltungen zusammen stecken.

Solche **„Prototype shields“** gibt es in unzähligen Ausführungen, Varianten und Größen für billiges oder erträgliches Geld bei Ebay.

Viel Spaß damit!