Teil 2

Lektion 25

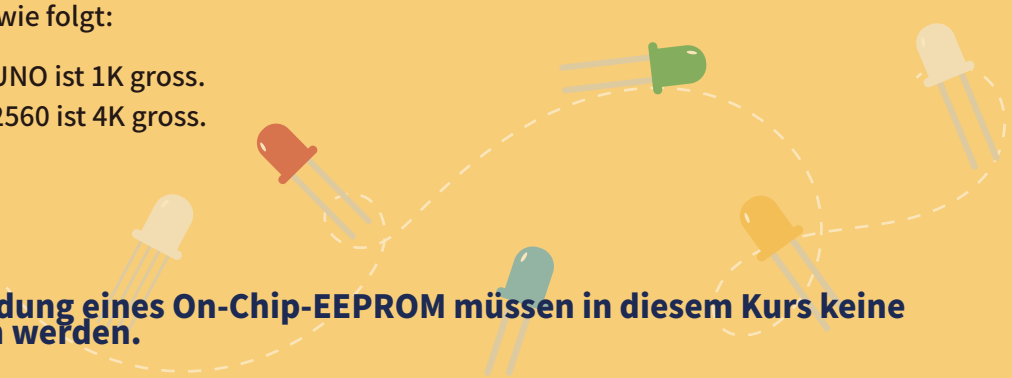
EEPROM

Überblick

EEPROM (Elektrisch löschbarer, programmierbarer Nur-Lese-Speicher),Ein Speicherchip, welcher Daten auch nach einer Unterbrechung der Stromversorgung nicht verliert. Kurz gesagt, wenn Arduino nach dem Ausschalten einige Parameter speichern soll, verwenden Sie das EEPROM.

Arduino-Controller sind mit AVR-Chips sowie mit EEPROM Chips ausgestattet. Die EEPROM-Größen gängiger Arduino-Controller lauten wie folgt:

Das EEPROM von Arduino UNO ist 1K gross.
Das EEPROM von Arduino 2560 ist 4K gross.



Benötigte Bauteile:

(1) x Elegoo Uno R3

Aufgrund der Verwendung eines On-Chip-EEPROM müssen in diesem Kurs keine Drähte angeschlossen werden.

Code

- Öffnen Sie nach der Verkabelung das Programm im Codeordner EEPROM und klicken Sie auf HOCHLADEN, um das Programm via Arduino DIE auf den Arduino hochzuladen. Weitere Informationen zum Hochladen von Programmen finden Sie in Lektion 5 von Teil 1, sollten Fehler auftreten.
- Das EEPROM von Arduino UNO kann nur 100.000 Mal gelöscht werden. Sie sollten also den Code, der die Funktion zum Lesen und Schreiben von Daten implementiert, nicht in loop() einfügen.
- In der Arduino EEPROM-Bibliothek beginnt die Adresse des EEPROM bei 0 und jede Adresse kann 1-Byte -Daten speichern. Wenn die Daten also größer als 1 Byte sind, müssen sie byteweise gelesen und geschrieben werden.
- Das EEPROM von Arduino UNO und Arduino Leonardo hat einen Speicherplatz von 1 KB = 1024 Byte und die entsprechende Adresse lautet 0 ~ 1023.
- Das EEPROM von Arduino Mega2560 verfügt über 4 KB = 4096 Byte Speicherplatz und die entsprechende Adresse lautet 0 ~ 4095.
- EEPROM.write(address,value);
  Funktion: Daten werden an die angegebene Adresse geschrieben;
- Parameters:
  Address: EEPROM-Adresse, Startadresse ist 0
  Im UNO liegt der Bereich zwischen 0 und 1023
  Im MEGA 2560 liegt der Bereich zwischen 0 und 4096
- Value: Daten, Typ 'Byte' oder 'Zeichen', der Bereich liegt zwischen 0 und 255. Dies bedeutet, dass eingehende Daten, die größer als dieser Bereich sind, abgeschnitten werden.
- EEPROM. Das Schreiben dauert jeweils 3 ms. Wenn das Programm das EEPROM weiterhin löscht, dauert es nicht lange, bis das EEPROM beschädigt ist. Achten Sie darauf, es nicht häufig zu löschen. Überlegen Sie es sich bitte genau, wann Sie diesen Speicher benutzen.

See Demo1 in the code for details:

```
char company[7] = {"elegoo"};
char company2[7] = {"0"};

 for(int i=0 ; i<6 ;i++ )
 {
 EEPROM.write( i,company[i]);
 }

  for(int i=0 ; i<6 ;i++ )
 {
  company2[i] = EEPROM.read(i);
 }
```

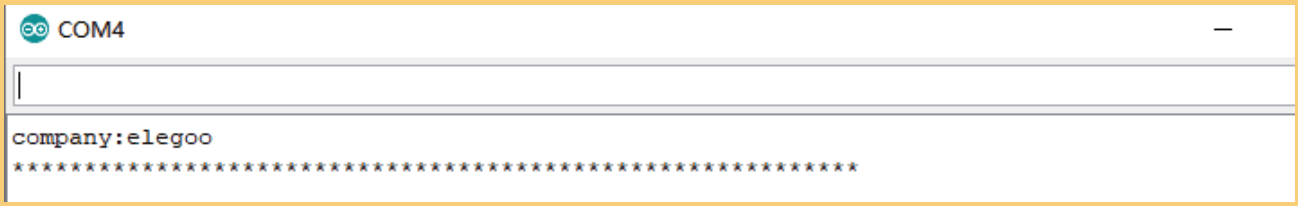
- EEPROM.read(address);
  Function: Daten von der angegebenen Adresse gelesen, und zwar jeweils 1 Byte-Daten. Wenn die angegebene Adresse keine Daten enthält, beträgt der Auslesewert 255.
- Grammar: EEPROM. read (address);
  Parameters:
  Address: Daten von der angegebenen Adresse einlesen, b eginnend mit 0.
  Return value: Gibt die gespeicherten Daten zurück, und zwar byte-weise.
- Nachdem Sie das Programm heruntergeladen haben, werden die Daten im EEPROM gespeichert. Dann können Sie versuchen, das Programm über EEPROM zu kommentieren. Schreibe es erneut. Laden Sie das Programm nochmals herunter und Sie können feststellen, dass die vom EEPROM gelesenen Daten mit den zuvor gespeicherten Daten übereinstimmen, auch wenn Sie das Arduino-Board zurücksetzen. Hiermit wird das dauerhafte Speichern von Daten beim Ausschalten des Boards realisiert.

```
void test1 (void)
{
 char company[7] = {"elegoo"};
 char company2[7] = {"0"};

// for(int i=0 ; i<6 ;i++ )
// {
// EEPROM.write(i,company[i]);
// }

  for(int i=0 ; i<6 ;i++ )
 {
  company2[i] = EEPROM.read(i);
 } }
```

- Klicken Sie auf die Schaltfläche “Serieller Monitor”, um den seriellen Monitor einzuschalten. Die Grundlagen des seriellen Monitors werden in Lektion 4 von Teil 2 ausführlich vorgestellt.



- Die Adresse des EEPROM beginnt bei 0 und jede Adresse kann 1 Byte-Daten speichern.
- **EEPROM. write** und **EEPROM. read** lesen kann nur mit Einzelbyte-Daten (wie Zeichen-, und Byte-Datentypen) realisiert werden.
- Wenn die eingehenden Daten (wie int, float usw.) größer als 1 Byte sind, werden sie abgeschnitten. Wenn die Daten also größer als 1 Byte sind, so werden diese byteweise gelesen und geschrieben.
- Als nächstes schreiben wir EEPROM Lese- und Schreibfunktionen für alle Typen
- In diesem Beispiel haben wir eine Funktions-Vorlage verwendet, um diese auf verschiedene Datentypen anzuwenden.
- **Siehe demo2 im Code für weitere Details.**

```
template<typename T>
void EEPROM_write(T data1,int address)
{
union data{
T my_type;
char charbuf[];
 } data2;
data2.my_type=data1;
for(int i=address ; i<sizeof(data1) ;i++ )
 {
 EEPROM.write(i , data2.charbuf[i]);
  }
  }
```

template<typename T>

- Der Vorlagen-Header <> kann entweder ein unbestimmter Datentyp sein, der durch den Typnamen definiert ist oder ein expliziter Datentyp.

- 'T' ist der generische Name der Kategorie. Die Datentypen, die Sie verwenden, sind int, char, float, double usw.
- Das obige Format ist die Funktionsvorlage
union: Mehrere verschiedene Arten von Variablen werden in derselben Speichereinheit gespeichert.Das heißt, mit Hilfe der Abdeckungs-Technologie decken sich mehrere Variablen gegenseitig ab.Diese Struktur, in der mehrere verschiedene Variablen zusammen einen Speicherabschnitt belegen, wird als “Vereinigung” bezeichnet.

```
union data{
T my_type;
char charbuf[];
} data2;
```

implementation of EEPROM_write

- Basierend auf der Art der “Vereinigung” speichern wir die Daten, die in das EEPROM geschrieben werden sollen, unter dem gleichen Typ von “my_type''. Da "my_type" und "charbuf []" denselben Speicher belegen, schreiben wir ein Array vom Char-Typ "charbuf" in Bytes in das EEPROM, was dem Schreiben von "my_type'to eeprom" entspricht, und EEPROM_read kann gleichermaßen erhalten werden.