Inhalt für das Mikrocontroller-Praktikum:

- 1× Koffer
- 1× Versuchplatine
- 1× Logic-Analyzer + USB-Kabel + Verbindungskabel
- 1× Kunststoffrohr
- 5× Pokerchips (weiß, schwarz, rot, blau, grün)
- 1× USB-Stick (Software, Treiber, Datenblätter,...)
- 1× Justierschraubendreher
- 10× Verbindungskabel
- 1× C-Cheatsheet

Sicherheitshinweise

- ☛ Die Platine darf nur mit dem USB-Port eines Computers verbunden werden.
- ☛ Weitere externe Hardware darf nur nach vorheriger Absprache mit den Betreuern angeschlossen werden.
- ☛ Eigene Lötarbeiten an der Platine sind untersagt.
- ☛ Die Praktikumsplatine ist trocken zu lagern und vor direkter Sonneneinstrahlung zu schützen.
- ☛ Defekte Platinen sind unverzüglich zu melden*.

Achtung! Kleinteile. Nicht geeignet für Kinder unter 3 Jahren.

* Es können immer wieder Bauteile kaputt gehen - in Ihrem eigenen Interesse sollten Sie uns dies in einem solchen Fall jedoch zeitnah melden, sodass wir defekte Teile austauschen und Ihnen schnellstmöglich eine funktionierende Platine zur Verfügung stellen können.

IMTEK

Lehrstuhl für Elektrische Mess- und Prüfverfahren

Prof. Dr. L. M. Reindl

Mikrocontroller-Praktikum

Lehrstuhl Elektrische Mess- und Prüfverfahren

C-Cheatsheet für das Arbeiten mit Mikrocontrollern

Version 1.3 - April 2015

UNI FREIBURG

IMTEK EMP Cheatsheet - Mikrocontroller-Praktikum

VARIABLEN

`int nameDerVariable;` // Initialisiert eine Integer-Variable. Mögliche Werte: -2^15 bis 2^15 - 1

`char nameDerVariable;` // Variable für ein Zeichen. Mögliche Werte: -128 bis +127

`unsigned char nameDerVariable;` // Ein (vorzeichenloses) Byte (0 bis 255)

Werte für **Variablen** werden in einem dieser zwei Formate angegeben:

Dezimal: 123
Hexadezimal: 0x7B

Auch möglich ist die Initialisierung mit Zeichen, was jedoch in der Regel nur bei Variablen des Typs `char` Sinn ergibt.

Darüber hinaus können Variablen auch als Array initialisiert werden:

```
int[6] zahlen; // Erstellt ein Array mit 6 Einträgen - d.h. es können
               // 6 int-Zahlen darin abgespeichert werden.
zahlen[0] = 1; // Erstes Element (0, nicht 1!) mit 1 initialisieren
zahlen[1] = 2;
zahlen[2] = 3;
zahlen[3] = 4;
zahlen[4] = 5;
zahlen[5] = 6; // Letztes Element mit 6 initialisieren (Größe 6 bedeutet 5 ist das größte Element!)
```

KONSTRUKTE

```
for (Startaussage; Bedingung; Aktion nach jedem Schleifendurchlauf) {
    // Anweisungen pro Schleifendurchlauf
    continue; // Aktuellen Schleifendurchlauf abbrechen
              // und mit nächsten fortfahren
}
```

```
while (Bedingung) {
  // Aktionen pro Schleifendurchlauf
  break; // Schleife abbrechen
}
```

`break` und `continue` können bei jedem Schleifentypen verwendet werden.

```
if (1. Bedingung) {
  // Aktionen, falls 1. Bedingung wahr ist.
}
else if (2. Bedingung) {
  // Aktionen, falls 1. Bedingung nicht wahr und 2. Bedingung wahr ist.
}
else {
  // Aktionen, falls 1. und 2. Bedingung nicht wahr sind.
}
```

```
switch (var) {
  case 0: <Anweisungen>;            // Wenn var == 0, mache etwas
  break;  // Der Rest in Switch wird übersprungen (wichtig bei default!)
  case 3: <Anweisungen>;            // Mache etwas anderes bei var == 3
  break;
  case 5: <Anweisungen> // ...(Man beachte die fehlende break-Anweisung!)
    ...
  default: <Eine default-Anweisung>;
           // Wenn var != 0 && var != 3, aber auch bei var == 5
           // =>(wegen dem fehlenden break!)
  break;   // Optional
}
```

OPERATOREN

Zuweisungsoperatoren: =

`zahl = 4;` // Initialisiert die Variable zahl und weist ihr den Wert 4 zu.

Vergleichsoperatoren: ==, >=, <=, !=

`1 == 1` ergibt 1.	`4 >= 2` ergibt 1.	`2 <= 8` ergibt 1.	`4 != 6` ergibt 1.
`7 == 3` ergibt 0.	`3 >= 7` ergibt 0.	`6 <= 2` ergibt 0.	`3 != 3` ergibt 0.

Mathematische Operatoren: +=, -=, /=, *=, ++, –, +, -, /, ∗, %

```
int zahl = 0;
zahl += 9;                      // zahl wird um 9 erhöht -> 9
zahl -= 3;                      // zahl wird um 3 verringert -> 6
zahl /= 2;                      // zahl wird durch 2 geteilt -> 3
zahl *= 4;                      // zahl wird mit 4 multipliziert -> 12
zahl++;                         // zahl wird um 1 erhöht -> 13
zahl--;                         // zahl wird um 1 verringert -> 12
zahl = (5 + 1 * 2 - 3) / 4;     // zahl wird auf 1 gesetzt (Punkt vor Strich beachten!)
zahl = 15 % 4;                  // zahl wird auf 3 gesetzt (Modulo, "Rest")
```

Binäre Operatoren: |, &, ^, ~, <<, >> (auch |=, &=, ^=)

Das binäre **ODER / OR**
`4 | 2` ergibt 6 denn: `4 = 0100 ; 2 = 0010 ; 4 | 2 = 0110`
`4 | 4` ergibt 4 denn: `4 = 0100 ; 4 = 0100 ; 4 | 4 = 0100`
`3 | 2` ergibt 3 denn: `3 = 0011 ; 2 = 0010 ; 3 | 2 = 0011`

Das binäre **UND / AND**
`4 & 2` ergibt 0 denn: `4 = 0100 ; 2 = 0010 ; 4 & 2 = 0000`
`4 & 4` ergibt 4 denn: `4 = 0100 ; 4 = 0100 ; 4 & 4 = 0100`
`3 & 2` ergibt 2 denn: `3 = 0011 ; 2 = 0010 ; 3 & 2 = 0010`

Das binäre **exklusives Oder / XOR**
`4 ^ 2` ergibt 6 denn: `4 = 0100 ; 2 = 0010 ; 4^2 = 0110`
`4 ^ 4` ergibt 0 denn: `4 = 0100 ; 4 = 0100 ; 4^4 = 0000`
`3 ^ 2` ergibt 1 denn: `3 = 0011 ; 2 = 0010 ; 3^2 = 0001`

Das binäre **NICHT / NOT**
`~0010` ergibt `1101`
`~1111` ergibt `0000`

Rechts- beziehungsweise **Linksshift**.
`4 << 1` ergibt 8 denn: `4 = 0100 4 << 1 = 1000` (Linksshift)
`3 << 2` ergibt 12 denn: `3 = 0011 3 << 1 = 1100` (Linksshift)
`3 >> 2` ergibt 0 denn: `3 = 0011 3 >> 2 = 0000` (Rechtsshift)
`3 >> 1` ergibt 1 denn: `3 = 0011 3 >> 1 = 0001` (Rechtsshift)

KOMMENTARE

Einzeilige Kommentare:

```
// Alles danach wird ignoriert
```

Mehrzeilige Kommentare:

```
/*
   Alles hierzwischen
   - auch Zeilenumbrüche -
   wird ignoriert
*/
```

FUNKTIONEN

```
int Funktionsname (char a, int b) // Definiert eine Funktion mit dem
                                  // Namen Funktionsname, den zwei
                                  // Übergabeparametern a  (Typ: char)
                                  // und b (Typ: int). Typ des
                                  // Rückgabewertes der Funktion: int.
{                                 // Beginn der eigentlichen Funktion
  // Hier der Code, der innerhalb der Funktion ausgeführt werden soll.
  int i = 14;
  i = i * 2;
  return i; // Rückgabe des Wertes und somit Beendigung der Funktion.
}           // Syntaktischer Abschluss der Funktion.

Funktionsname( a, 2);             // Exemplarischer Aufruf der Funktion
int y = Funktionsname(14, 0x14); // Alternativer Aufruf bei dem die
                                  // Rückgabe in y gespeichert wird
```