B Fortgeschrittene Arduino-Programmierung

Im Grunde ist die Ardunio-Programmiersprache nichts anderes als C++, aber sie hat einige Einschränkungen und sie verwendet eine spezielle Umgebung. In diesem Anhang lernen Sie die Einschränkungen kennen. Zusätzlich finden Sie einen kurzen Abschnitt, der verrät, wie Bit-Operationen funktionieren, denn Sie benötigen sie häufig, wenn Sie mit Sensoren und anderen Geräten arbeiten.

B.1 Die Arduino-Programmiersprache

Die ersten Programme, die Sie für Arduino schreiben, scheinen in einer besonderen Arduino-Sprache geschrieben worden zu sein, aber das stimmt nicht. Um Arduino zu programmieren, verwenden Sie normalerweise das gute alte C/C++. Leider versteht der Arduino keinen C- oder C++-Code, weshalb Sie ihn auf Ihrem PC oder Mac in Maschinensprache für den Microcontroller von Arduino kompilieren müssen. Dieser Vorgang wird Cross-Compiling genannt und ist die übliche Vorgehensweise, um ausführbare Software für Microcontroller zu kompilieren. Dabei bearbeiten und kompilieren Sie die Software auf Ihrem PC und übertragen dann den Maschinencode an den Microcontroller.

Im Falle des Arduino handelt es sich bei diesen Microcontrollern um Mitglieder der AVR-Produktfamilie einer Firma namens Atmel. Um die Softwareentwicklung für AVR-Microcontroller so einfach wie möglich zu machen, hat Atmel eine ganze Batterie von Werkzeugen auf Basis der GNU Compiler-Tools entwickelt. Die Tools verhalten sich wie das Original, sind aber optimiert für die Codeerzeugung für die Atmel-Microcontroller.

Für beinahe alle GNU-Entwicklungstools wie *gcc*, *ld* oder *as* gibt es eine AVRVariante: *avr-gcc*, *avr-ld* usw. Sie finden Sie im Verzeichnis *hardware/tools/bin* der Arduino-IDE.

Im Wesentlichen ist die IDE eine grafische Oberfläche, die es Ihnen erspart, die Kommandozeilenbefehle direkt einzugeben. Immer, wenn Sie ein Programm kompilieren oder mit der IDE hochladen, delegiert sie die Arbeit an die AVR-Tools. Als seriöser Softwareentwickler sollten Sie eine ausführlichere Ausgabe einstellen, so dass Sie alle aufgerufenen Kommandozeilenbefehle sehen können. Richten Sie die ausführliche Ausgabe wie in Abschnitt 2.3, *Voreinstellungen ändern*, beschrieben sowohl für die Kompilierung als auch für das Hochladen ein. Laden Sie dann unser LED-Blinkprogramm und kompilieren Sie es. Die Ausgabe im Meldungsbereich sollte wie in Abbildung 2–4 aussehen.

Die eingeblendeten Kommandozeilenbefehle sehen erst etwas eigenartig aus, wegen der vielen temporären Dateien. Sie sollten aber noch immer in der Lage sein, jeden Schritt beim Kompilieren und Linken zu erkennen, der notwendig ist, um Ihr Blinkprogramm zu erstellen. Das ist das Wichtigste, was das Arduino-Team getan hat: Man hat alle diese nervigen kleinen Schritte prima hinter der IDE versteckt, so dass auch Menschen ohne Erfahrung in der Softwareentwicklung einen Arduino programmieren können. Für Programmierer ist es sinnvoll, im ausführlichen Modus zu arbeiten, da er die beste Möglichkeit ist, alle AVR-Tools kennenzulernen und sie in Aktion zu sehen.

Laden Sie das Programm jetzt auf den Arduino hoch, um *avrdude* in Aktion zu erleben. Dieses Tool ist zuständig für das Laden des Codes in den Arduino und kann auch für die Programmierung anderer Geräte verwendet werden. Interessanterweise können Sie mit den AVR-Tools die Arduino-IDE auch für andere NichtArduino-Projekte verwenden wie Meggy Jr.[1].

»Halt!«, werden Sie jetzt sagen, »ich bin C++-Programmierer und ich vermisse die *main*-Funktion!« Sie haben Recht: Dies ist ein weiterer Unterschied zwischen Arduino-Programmierung und regulärem C++-Code. Wenn Sie für Arduino programmieren, definieren Sie *main* nicht selbst, weil es bereits in den Bibliotheken für Arduino-Entwickler definiert wurde. Wie Sie vielleicht erraten haben, ruft *main* zuerst *setup* auf und führt die *loop*-Funktion in einer Schleife aus. Seit Arduino 1.0 ruft sie am Ende der *loop*-Funktion auch *serialEvent* auf.

Es gibt noch weitere Einschränkungen, wenn Sie AVR-Microcontroller mit C++ programmieren:[2]

- Sie können die Standard Template Library (STL) nicht benutzen, da sie für die kleinen AVR-Microcontroller viel zu groß ist.
- Ausnahmebehandlung wird nicht unterstützt. Deshalb sehen Sie den Schalter *-fno-exceptions* häufig, wenn der *avr-gcc*-Compiler aktiviert wird.
- Dynamisches Speichermanagement mit *new* und *delete* wird momentan nicht unterstützt.

Zusätzlich sollten Sie auf die Performanz achten. C++ erzeugt beispielsweise automatisch eine Menge Funktionen im Hintergrund (Kopier-Konstruktoren, Zuweisungsoperatoren usw.), die selten auf dem Arduino gebraucht werden. Trotz dieser Einschränkungen unterstützt Arduino eine leistungsfähige Untermenge der Programmiersprache C++. Es gibt also keine Ausreden für nachlässiges Kodieren!

B.2 Bit-Operationen

Im Emdedded-Computing müssen Sie oft mit Bits arbeiten. Zum Beispiel müssen Sie einzelne Bits lesen, um an Sensordaten zu kommen. In anderen Fällen müssen Sie Bits setzen, um ein Gerät in einen bestimmten Status zu versetzen oder damit es eine Aktion ausführt.

Zur Bit-Manipulation benötigen Sie nur wenige Operationen. Die einfachste ist die Operation NOT, die ein Bit invertiert. Sie macht aus 0 eine 1 und umgekehrt. Die meisten Programmiersprachen implementieren NOT mit dem ~-Operator.

```
int x = 42; // Binär ist das 101010
int y = ~x; // y == 010101
```

Zusätzlich finden Sie drei Binäroperationen namens AND, OR und XOR (Exklusiv-Oder). Die meisten Programmiersprachen nennen die dazugehörigen Operatoren &, |, und ^, und ihre Definitionen sind wie folgt:

a	b	a AND b a & b	a OR b a | b	a XOR b a ^ b
0	0	0	0	0
1	0	0	1	1
0	1	0	1	1
1	1	1	1	0

Mit diesen Operatoren ist es möglich, die Bits einer Zahl zu maskieren. Zum Beispiel können Sie bestimmte Bits extrahieren. Wenn Sie an den unteren beiden Bits einer Zahl interessiert sind, machen Sie das wie folgt:

```
Int x = 42;       // binär ist das 101010
Int y = x & 0x03; // y == 2 == B10
```

Sie können ein oder mehrere Bits in einer Zahl mit der OR-Operation setzen. Der folgende Code setzt das fünfte Bit in x, unabhängig davon, ob es 0 oder 1 ist.

```
int x = 42;       // binär ist das 101010
int y = x | 0x10; // y == 58 == B111010
```

Die Verschiebe-Operatoren << und >> lassen Sie Bits in eine bestimmte Position verschieben, bevor Sie mit ihnen arbeiten. Der erste verschiebt Bits nach links und der zweite nach rechts:

```
int x = 42;     // binär ist das 101010
int y = x << 1; // y == 84 == B1010100
int z = x >> 2; // z == 10 == B1010
```

Verschiebeoperationen scheinen intuitiv zu sein, Sie müssen jedoch vorsichtig sein, wenn Sie vorzeichenbehaftete Werte verschieben.[3] Auch wenn sie ähnlich aussehen, sind Binäroperatoren nicht dasselbe, wie Boolesche. Boolesche Operatoren wie && und || funktionieren nicht auf Bitebene. Sie implementieren boolesche Algebra[4].

Anfänger haben oft Angst vor Bitoperationen, aber dazu besteht kein Grund. Microcontroller arbeiten auf Bitebene, weshalb Sie eine Möglichkeiten brauchen, um den Bits Ihren Willen aufzuzwingen. Das erfordert nur etwas Übung, ist aber keine höhere Mathematik.