Mikrocontroller - *objektorientiert programmieren*

Was haben Mikrocontroller und OOP miteinander zu tun ? Bislang wurden Mikrocontroller in Assembler, Basic, Pascal, zumeist aber in C programmiert. Der Firma Tasking, welche sich auf Compiler spezialisiert hat, war das offensichtlich nicht genug, und so wurde einer der ersten C++ - Compiler für Mikrocontroller geboren.

Christian Perschl DSK340//song

Damit können die Säulen von OOP - Datenkapselung, Vererbung und Polymorphie und die damit verbundene Wiederverwendbarkeit - auch bei der Programmierung von Mikrocontrollern ihre Mächtigkeit unter Beweis stellen. So könnte man externe Einheiten, wie z.B. Sensoren, Motoren, Displays oder Schalter in Klassen abstrahieren. Damit erreicht man einerseits einen kontrollierten und überschaubaren Zugriff auf diese Einheiten, und andererseits können die so erstellten Routinen leicht wiederverwendet werden. Aber auch die Implementierung von Anwendungsobjekten ist sinnvoll. Im Programmbeispiel werden solche Anwendungsobjekte erzeugt.

Beim Tasking C++-Compiler für die Siemens C166-Familie ist kein direkter Zugriff auf die Special Function Register und die Peripherie möglich. Es müssen zuerst Ansi-C Funktionen, welche die Peripherie steuern, implementiert werden - z.B. als Bibliothek. Diese werden dann in den Methoden der einzelnen Klassen aufgerufen. Dies ist zwar offensichtlich als weiterer Abstrahierungsgrad gedacht, beeinträchtigt aber einerseits die Performance und andererseits die Fähigkeit, rasch Applikationen zu erstellen. Beispiel: Eine fiktive Funktion `ADWert_lesen()` macht nichts anderes, als den Wert des AD-Wandler-Ergebnisregisters zurückzugeben. So entstehen womöglich eine Reihe von Funktionen mit nur einem Befehl, und durch mehrmaliges Umleiten der Rückgabe wird das Programm auch nicht schneller.

Ein erstes C++ - MC - Programm

Nichtsdestoweniger ist es eine reizvolle Aufgabe, eine erste objektorientierte Applikation für einen Mikrocontroller zu schreiben. Aufgabenstellung ist es, einen mehrstimmigen Musikgenerator mit Liedern, die in der bekannten Notation des Basic-Befehles PLAY vorliegen, in C++ für den C167CR zu implementieren.

Das Format

Hier möchte ich kurz auf das Format des Eingabestrings eingehen. Folgende Zeichen werden von der Einleseroutine akzeptiert:

C, D, E , F, G, A, B	Spiele die entsprechenden Noten
+,-	erhöht bzw. erniedrigt die vorangegangene Note um einen Halbton, entspricht einem Kreuz (#) bzw. b bei Noten
Lx	Stellt die Länge für die folgenden Noten ein. z.B. x=1 Ganze Note, x=2 Halbe Note, x=4 Viertel Note...
.	Ein Punkt verlängert die vorangegangene Note um die Hälfte ihrer „ursprünglichen“ Länge
Ox	Wählt die Oktave aus, x=0..6
Px	Pause mit der Länge x (wie bei Lx)

Beispiel

O4L8CDEFL4GGL8AAAAL2GL8AAAAL2GL8FFFFL4EEL8DDDDL4C.P8

spielt ein bekanntes Kinderlied

Die Klasse song

Jedes Lied wird als Instanz der Klasse `song` angelegt. Die Klasse `song` enthält die Methoden `play` zum Abspielen und `einlesen` zum Einlesen des Eingabestrings, welcher als Eigenschaft `song_string` der Klasse implementiert ist.

Die Klasse `song` ist folgendermaßen aufgebaut:

```
class song
{
public:
  song(char *liedname,char *lied); /* 1. Konstruktor :
                                      Liedname und Eingabestring */
  song(char *lied);      /* 2. Konstruktor: Nur Liedname */
  int status(void);      /* Gibt den Fortschritt
                            beim Abspielen aus */
  void start(void);      /* Initialisiert alle Parameter
                            zum Abspielen */
  void next(void);       /* Spielt nächste Note */
  void stop(void);       /* Beendet das Abspielen */
  void play(void);       /* Spielt das Lied ab
                            und gibt Status aus */
  int channel;           /* Ausgabe Kanal (0-3) */
  char *name;            /* Liedname */
private:
  void einlesen (void);  /* Liest Eingabestring ein und
                            setzt note, laenge..*/
  int tempo;             /* aktuelles tempo in bpm */
  int note;              /* aktuelle note
                            (0-11 =C-B, 12=Pause) */
  int oktave;            /* aktuelle oktave (0-6) */
  int laenge;            /* aktuelle notenlänge */
  int song_counter;      /* Aktuelle Position
                            im Eingabestring */
  char *song_string;     /* Eingabestring */
  int oldlaenge;         /* Sicherungsvariable für oldlaenge */
};
```

Die Methode `play` spielt das gesamte Lied ab und verwendet dazu die Methoden `start`, `next` und `stop`.

Zur Tonerzeugung wird die 4-Kanal PWM-Einheit verwendet, für die Tonlänge Timer. Welcher PWM-Kanal und welcher Timer verwendet wird, hängt von der Eigenschaft `channel` ab.

Die Klasse msong

Die Klasse `song` kann ein Lied nur einstimmig abspielen. Um nun mehrere Lieder/ Stimmen gleichzeitig abzuspielen, werden nun die einzelnen Stimmen als `song` - Objekte angelegt. Die Klasse `msong` enthält Zeiger auf 4 Objekte der Klasse `song`, welche bei der Initialisierung des `msong`-Objektes übergeben werden.

Die Methode `play` der Klasse `msong` macht jetzt nichts anderes, als die Methoden `start`, `next` und `stop` aller 4 Objekte gleichzeitig durchzuführen.

Instanzierung

```
song Carmen1(“.......“);  /* erste Stimme */
song Carmen2(“.......“);  /* zweite Stimme */
song Carmen3(“.......“); /* dritte Stimme */
song Carmen4(“.......“); /* vierte Stimme */

msong Carmen(Carmen1, Carmen2, Carmen3, Carmen4);
```

Im Konstruktor der Klasse `msong` werden die Objekte als Referenzen angegeben und die Adressen der Objekte den Pointern `s1-s4` zugewiesen:

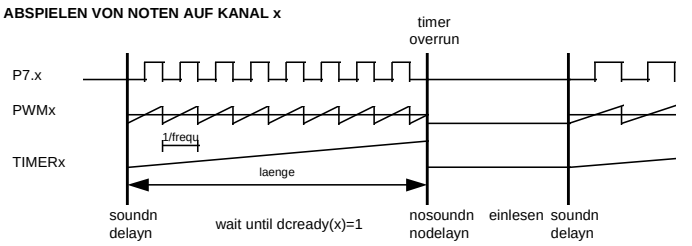
```
class msong
{
  public:
   msong(song &s1,song &s2,song &s3,song &s4);
   void play(void);
  private:
   song *s1,*s2,*s3,*s4;
};
```

Implementierung

Die Frequenzen der einzelnen Noten werden mit Hilfe der 4-Kanal-PWM - Einheit des C167CR erzeugt. Hierbei wird während des Abspielens keine CPU-Zeit verbraucht, die PWM-Kanäle werden zu Beginn des Tones lediglich richtig initialisiert, gestartet und am Ende des Tones angehalten. Durch Änderung des Puls-Verhältnisses läßt sich auch die Klangfarbe variieren.

Die Tondauer wird durch die ersten 4 Timer-Einheiten (T0, T1, T2, T3) des C167CR festgelegt. Jeder Timer läuft für sich als sog. delay channel, und in der Interrupt Service Routine des Timers wird das globale Bit `dcx` (Delay Channel Bit) gesetzt. Die Methode `play` frägt die 4 Delay Channel Bits im Polling-Betrieb ab, und sobald ein Delay Channel „abgelaufen" ist - die Tonlänge wurde erreicht -, wird die nächste Note mit der Methode `next` eingelesen.

Die nächste Abbildung zeigt das Ausgangssignal, das Programmverhalten sowie Timer- und PWM-Einheiten:



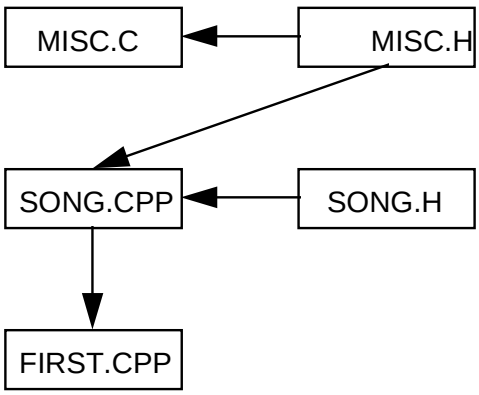
Die Funktion `soundn` setzt die Werte des PWM-Kanales x auf die Frequenz der aktuellen Note. Mit `nosoundn` wird der PWM-Kanal x deaktiviert.

Die Funktion `delayn` initialisiert den Timer x auf die in ms angegebene Notenlänge laenge. `Nodelayn` deaktiviert den Timer x.

Mit der Methode `einlesen` der Klasse `song` wird nach dem Abspielen einer Note die nächste geladen und die Werte `note`, `laenge`, `oktave`, `tempo` aktualisiert.

Implementierungsschichten

Die Applikation ist in mehrere Ebenen gegliedert: `FIRST.CPP` ist das Hauptprogramm, in dem nur die DIP-Schalterstellung abgefragt und dann das entsprechende Lied gespielt wird. In `SONG.CPP` sind die Klassen `song` und `msong` implementiert. `MISC.C` enthält alle erforderlichen Routinen zur Ansteuerung der Peripherie (dies ist in C++ direkt nicht möglich) und allgemeine Funktionen.



- Allgemeine ANSI-C Funktionen zur Ansteuerung der OnChip-Peripherie
- Klassendefinition und Implementierung von `song` und `msong`
- Hauptprogramm

Dateiliste

Allgemeine Funktionen für MC

`fehler.c`	Interruptvektoradressen für Laufzeitfehler sinnvoll beschreiben
`serio.c`	Vom Compilerhersteller mitgelieferte Datei (für `printf` erforderlich)
`serio.h`	Vom Compilerhersteller mitgelieferte Datei (für `printf` erforderlich)
`_doprint.c`	Vom Compilerhersteller mitgelieferte Datei (für `printf` erforderlich)
`cstart.asm`	Vom Compilerhersteller mitgelieferte Datei. Programm, welches vor `main()` aufgerufen wird und den MC initialisiert

MC-Programme

`first.cpp`	Hauptprogramm
`song.cpp`	Methoden - Implementierung der Klassen `song` und `msong`
`song.h`	Klassendefinition von `song` und `msong`
`misc.c`	Ansi-C Funktionen zur Ansteuerung der Peripherie
`misc.h`	Extern-Schnittstelle der allgemeinen Funktionen

Steuerdateien

`makefile`	Steuerdatei für mk166 (Generierung des ausführbaren MC-Programmes)
`cmd_link`	Kommandodatei für den Linker
`cmd_loc.e_e`	Kommandodatei für den Locater

PC-Programmdateien

`term.cpp`	Source-Code für PC-Applikation (Visualisierung der Spieldauer)
`term.exe`	Ausführbare PC-Applikation

Dokumentation

`oop.doc`	Diese Datei (im Winword 6.0 - Format)
`timer.ppt`	Zeitdiagramm
`dateien.ppt`	Implementierungsschichten

HOTLINE *„Ford Hotline, was kann ich für Sie tun?"*
KUNDE *„Ihre Autos sind Mist!"*
HOTLINE *„Was ist passiert?"*
KUNDE *„Es ist kaputt, das ist passiert!"*
HOTLINE *„Was genau haben Sie getan?"*
KUNDE *„Ich wollte schneller fahren. Also habe ich das Gaspedal bis zum Boden durchgedrückt. Eine Weile ging alles gut aber jetzt ist der Wagen kaputt und läßt sich nicht mehr starten!"*
HOTLINE *„Es liegt in Ihrer Verantwortung wenn Sie unser Produkt mißbrauchen. Was erwarten Sie nun von uns?"*
KUNDE *„Ich verlange, daß Sie mir das neueste Modell schicken, das nicht mehr kaputt geht!"*