Embedded C/C++

Einige Hinweise und Tipps

Peter Ulbrich, Martin Hoffmann

Friedrich-Alexander-Universität Erlangen-Nürnberg
Lehrstuhl Informatik 4 (Verteilte Systeme und Betriebssysteme)
www4.informatik.uni-erlangen.de

30. Oktober 2011

Präprozessormakros

- Register entspricht einer Stelle im Speicherbereich
- Register hat eine Adresse
- Interpretiere Adresse als Zeiger

```
#define TIER *((volatile unsigned char*)0xff90)

unsigned char a = TIER;
TIER = 5;
TIER |= 0x80;
```

- `volatile` schützt vor übereifrigem Compiler

Präprozessormakros

- eleganter: `union` (aber nicht wirklich standardkonform)

```
union tier {
  volatile unsigned char val;
  struct {
    volatile unsigned char ICIAE : 1;
    ...
  } bits;
};

#define TIER *((union tier*)0xff90)

unsigned char a = TIER.val;
TIER.bits.ICIAE = 1;
TIER.val |= 0x85;
```

Operatoren

- Überladen des
  - Zuweisungsoperators
  - Typumwandlungsoperators (für Referenzen)

```
template < typename TYPE, int ADDR > class MemMap {
public:
  TYPE operator=(TYPE val) const {
    *((volatile TYPE*)ADDR) = val; return val;
  }
  operator TYPE () const {
    return *((volatile TYPE*)ADDR);
  }
  operator TYPE& () const {
    return *((volatile TYPE*)ADDR);
  }
};
```


Operatoren

- Verwendung

```
MemMap< unsigned char,0xff90 > TIER;

unsigned char a = TIER;
TIER = 0x80;
TIER |= 0x40;
TIER &= ~0x80;
```


Warum Assembler?

- Effizienz??? $\rightsquigarrow$ in unserem Fall eher nicht

☞ Der Compiler unterstützt nicht alle Instruktionen der CPU.

☞ Manches lässt sich in einer Hochsprache nicht formulieren.

Assemblerdateien

my_add.s

```
_my_add:
  mov 8(%esp) ,%eax
  mov 12(%esp),%edx
  add %edx,%eax
  ret
```

my_source.c

```
unsigned int my_add(unsigned int,unsigned int);
int main() {
  unsigned int a = 3,b = 5,c = 0;
  c = my_add(a,b);
  return 0;
}
```

Inline-Assembler

- kleine Assembler-Abschnitte: Aufwand einer eigenen Datei groß

Inline-Assembler

```
int main() {
  unsigned int a = 3,b = 5,c = 0;
  __asm__ ("mov %2,%eax;"
           "mov %3,%edx;"
           "add %edx,%eax;"
           "mov %eax,%1"
           : "=m" (c)
           : "m" (a),"m" (b)
           : "eax","edx");
  return 0;
}
```


Vorsicht: Ganzzahlarithmetik

Divisionen

```
unsigned int a = 99, b = 100;
unsigned int c = a / b; // c == 0 !
```

Multiplikationen

```
unsigned short a = 1000, b = 100;
unsigned short c = a * b; // c != 100000 !
                          // c == 34464 !
```

Additionen und Subtraktionen

```
unsigned short a = 0xffff;
a++;   // a != 0x10000 !
       // a == 0 !
```


Gleitkommaarithmetik

- Viele Prozessoren unterstützen keine Gleitkommazahlen
- Abbildung auf Ganzzahlarithmetik
- Extrem teuer
- Braucht man für ein Echtzeitbetriebssystem (eigentlich) nicht

Keine dynamische Speicherverwaltung!

- kein `malloc/free`
- kein `new/delete`
- keine Speicherlöcher!
- Speicher wird angefordert:
  - dynamisch auf dem Stack
  - statisch im Datensegment (globale Variablen)

Globale Variablen

- Problem:

```
A a;
B b;
C c;
```

In welcher Reihenfolge werden die Konstruktoren aufgerufen?

- Lösung: `__attribute__((init_priority(x)))`

```
A a __attribute__((init_priority(1000))); // 1.
B b __attribute__((init_priority(2000))); // 2.
C c __attribute__((init_priority(3000))); // 3.
```

- GCC only - im Standard nicht spezifiziert