ALG-Seminar Sensornetze

C++ in Eingebetteten Systemen

Tobias Baumgartner

Braunschweig Institute of Technology

July 16th, 2009

Outline

1. Mikrocontrollerprogrammierung
   - Alignment-Probleme
   - Interrupt-Kontext
2. Effizientes C++
   - Virtuelle Methoden
   - Das Schlüsselwort `inline`
   - Auch Kleinvieh macht zum Teil viel Mist
3. Kleinere Fallstricke...
4. Weiterführende Literatur

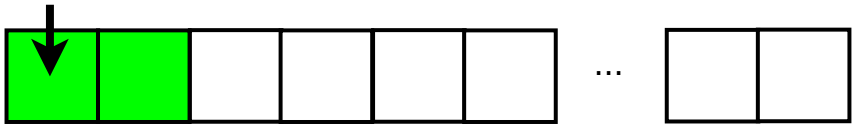
Outline

1. Mikrocontrollerprogrammierung
   - Alignment-Probleme
   - Interrupt-Kontext
2. Effizientes C++
   - Virtuelle Methoden
   - Das Schlüsselwort `inline`
   - Auch Kleinvieh macht zum Teil viel Mist
3. Kleinere Fallstricke...
4. Weiterführende Literatur

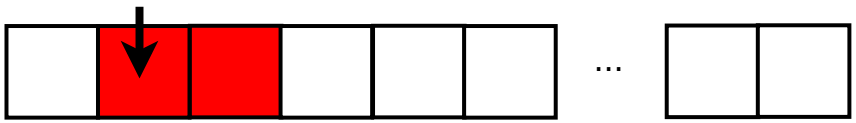
Problem

char buffer[BUF_LEN];

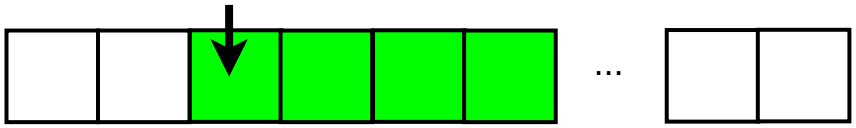
uint16_t = *(uint16_t*)buf; // MSP, Jennic



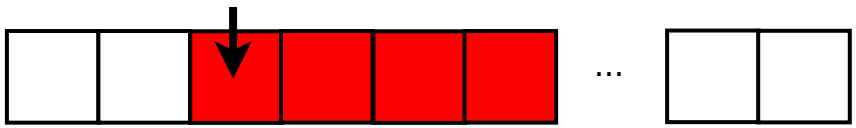
uint16_t = *(uint16_t*)(buf + 1); // MSP, Jennic



uint32_t = *(uint32_t*)(buf + 2); // MSP



uint32_t = *(uint32_t*)(buf + 2); // Jennic



Speicherzugriff via structs

1. Struct

```
1  struct TestStruct
2  { uint8 elem1; uint16 elem2; };
3
4  TestStruct *st_even = (TestStruct*)buf;
5  TestStruct *st_odd = (TestStruct*)(buf + 1);
6  os_.debug("%x, %x", st_even->elem1, st_even->elem2);
7  os_.debug("%x, %x", st_odd->elem1, st_odd->elem2); // Fatal!
```

2. Packed Struct

```
1  struct TestStruct2
2  { uint8 elem1; uint16 elem2; } __attribute__((packed));
3
4  TestStruct2 *st2_even = (TestStruct2*)buf;
5  TestStruct2 *st2_odd = (TestStruct2*)(buf + 1);
6  os_.debug("%x, %x", st2_even->elem1, st2_even->elem2);
7  os_.debug("%x, %x", st2_odd->elem1, st2_odd->elem2);
```

3. Zusammensetzen

```
1  os_.debug("%x, %x", buf[0], (uint16)(buf[1] << 8 | buf[2]));
2  os_.debug("%x, %x", buf[1], (uint16)(buf[2] << 8 | buf[3]));
```

Codesize Speicherzugriffe

	Struct	Packed Struct	Zusammensetzen
Codesize	176	188	188

- Wisebed-SVN:
  `https://www.ibr.cs.tu-bs.de/svn/wisebed/trunk/isense_apps/alignment-test`
- `make AlignmentSize` kompiliert `src/minitest.cpp` und zeigt Code-Grösse an
- `make JN5139R1` erstellt Beispielapplikation `src/alignment-test.cpp` für iSense, die auf Knoten getestet werden kann

Outline

1. Mikrocontrollerprogrammierung
   - Alignment-Probleme
   - Interrupt-Kontext
2. Effizientes C++
   - Virtuelle Methoden
   - Das Schlüsselwort `inline`
   - Auch Kleinvieh macht zum Teil viel Mist
3. Kleinere Fallstricke...
4. Weiterführende Literatur

Interrupt- und User-Kontext

- Interrupt-Kontext
  - Ausgelöst durch Ereignisse wie eingehende Bytes über UART oder Radio, Timer, Fehler (unaligned mem access), ...
  - Beispiel: `iSense::TimeoutHandler`
  - Während der Bearbeitung blockiert man i.d.R. andere Interrupts
  - Deshalb: **Schnell** sein, d.h. keine Debug-Ausgaben (oder wenn, dann nur einzelne Zeichen)
  - Bewusstsein, was hinter aufgerufenen Funktionen steckt (z.B. `new`/`malloc` in iSense)
- User-Kontext
  - Berechnungsintensivere Aufgaben
  - Unterbrechung durch Interrupts
  - Beispiel: `iSense::Task`
  - **Aber**: Normalerweise kein Multi-Threading, deshalb auch kein blockierendes Warten

Outline

1. Mikrocontrollerprogrammierung
   - Alignment-Probleme
   - Interrupt-Kontext
2. Effizientes C++
   - Virtuelle Methoden
   - Das Schlüsselwort `inline`
   - Auch Kleinvieh macht zum Teil viel Mist
3. Kleinere Fallstricke...
4. Weiterführende Literatur

Outline

1. Mikrocontrollerprogrammierung
   - Alignment-Probleme
   - Interrupt-Kontext
2. Effizientes C++
   - Virtuelle Methoden
   - Das Schlüsselwort `inline`
   - Auch Kleinvieh macht zum Teil viel Mist
3. Kleinere Fallstricke...
4. Weiterführende Literatur

Anwendung von `virtual`

- Überschreiben von Methoden in Basisklasse

```
1 class A { ... virtual void foo(); ... }
2 class B { ... virtual void foo(); ... }
3
4 B b;
5 A *a = &b;
6 a->foo();
```

Overhead von `virtual`

1. Pro virtuelle Methode ein zusätzlicher Pointer
2. Zusätzlicher indirekter Methodenaufruf (o.g. Pointer)
3. Compiler kann nicht optimieren (kein `inline`)
4. Mini-Test in

   `https://svn.itm.uni-luebeck.de/wisebed/wiselib/trunk/sandbox/inline-eval`

	virtuelle Methode	*normale* Methode
Codesize	123	68

(per Hand einmal mit virtueller Methode in `BaseClass` kompiliert, einmal ohne `virtual`)

Outline

1. Mikrocontrollerprogrammierung
   - Alignment-Probleme
   - Interrupt-Kontext
2. Effizientes C++
   - Virtuelle Methoden
   - Das Schlüsselwort `inline`
   - Auch Kleinvieh macht zum Teil viel Mist
3. Kleinere Fallstricke...
4. Weiterführende Literatur

Wann wird `inline` angewendet?

- Automatisch, wenn eine Methode in der Klassendefinition implementiert wird

```
1 class A {
2    ...
3    void autoinline_method()
4    { i++;  }
5 }
```

- Explizite Angabe des Schlüsselwortes `inline` (Implementierung muss aber im Header geschehen)

```
1 class A {
2    ...
3    inline void inline_method();
4 }
5 ...
6 void A::inline_method() { i++ };
```

- **Aber:** Nur eine *Empfehlung* an den Compiler - zu *grosse* Methoden werden nicht *geinlined*

`inline`-Limit

- Intern benutzt der Compiler ein `inline`-Limit (`-finline-limit=N`)
- Bei manchen Compilern wird das `inline-limit` bei Verwendung von `-Os` sehr klein angesetzt (`ba-elf-g++`: < 36)
- Daher wird `inline` z.B. nicht auf Methoden angewendet, in denen eine virtuelle Methode aufgerufen wird (zumindest nicht mit dem `ba-elf-g++ (GCC) 4.2.1`)
- Mini-Test in

  `https://svn.itm.uni-luebeck.de/wisebed/wiselib/trunk/sandbox/inline-eval`

inline-Limit

```
#######################################
# Compile with -O2
#######################################
ba-elf-g++ -O2 -c inline-test.cpp -o inline-test.o
ba-elf-nm inline-test.o | grep value
00000000 T __ZN9BaseClass5valueEv
ba-elf-size inline-test.o
   text    data     bss     dec     hex filename
    123       0       0     123      7b inline-test.o
#######################################
# Compile with -Os
#######################################
ba-elf-g++ -Os -c inline-test.cpp -o inline-test.o
ba-elf-nm inline-test.o | grep value
00000000 W __ZN15StaticTestClass6value2EP9BaseClass
00000000 T __ZN9BaseClass5valueEv
ba-elf-size inline-test.o
   text    data     bss     dec     hex filename
    159       0       0     159      9f inline-test.o
#######################################
# Compile with -Os -finline-limit=120
#######################################
ba-elf-g++ -g -Os -finline-limit=120 -c inline-test.cpp -o inline-test.o
ba-elf-nm inline-test.o | grep value
00000000 T __ZN9BaseClass5valueEv
ba-elf-size inline-test.o
   text    data     bss     dec     hex filename
    123       0       0     123      7b inline-test.o
```

Outline

1. Mikrocontrollerprogrammierung
   - Alignment-Probleme
   - Interrupt-Kontext
2. Effizientes C++
   - Virtuelle Methoden
   - Das Schlüsselwort `inline`
   - Auch Kleinvieh macht zum Teil viel Mist
3. Kleinere Fallstricke...
4. Weiterführende Literatur

Vielfache Methodenaufrufe

- Viele Methodenaufrufe

```
1 uart.put( 0x01 );
2 uart.put( 0x02 );
3 uart.put( 0x03 );
4 uart.put( 0x04 );
5 uart.put( 0x05 );
```

- Ein Methodenaufruf

```
1 char x[] = { 0x01, 0x02, 0x03, 0x04, 0x05 };
2 uart.write( x, sizeof(x) );
```

- Codesize-Vergleich

	Viele Puts	Ein Write
Codesize	184	156

int vs. uint8_t & Co

```
1   // Fall 1:
2   char add( char x, char y )
3   { return x + y; }
4   // Fall 2:
5   int add( int x, int y )
6   { return x + y; }
7
8   int main()
9   {
10    int j = 1;
11    for (int i = 0; i < 10; i++)
12      j = add( i, j );
13    return j;
14  }
```

```
1   ba-elf-g++ -O0 -c test.cpp -o test.o
2   ba-elf-size test.o
3   // Fall 1:
4   text    data    bss    dec    hex    filename
5   180     0       0      180    b4     test.o
6   // Fall 2:
7   text    data    bss    dec    hex    filename
8   136     0       0      136    88     test.o
```

Outline

1. Mikrocontrollerprogrammierung
   - Alignment-Probleme
   - Interrupt-Kontext
2. Effizientes C++
   - Virtuelle Methoden
   - Das Schlüsselwort `inline`
   - Auch Kleinvieh macht zum Teil viel Mist
3. Kleinere Fallstricke...
4. Weiterführende Literatur

Bedeutung von `static`

- Schlüsselwort `static` innerhalb Klassen

```
1  class XYZ {
2    static void tuwas() {...};
3  }
4  ...
5  XYZ::tuwas();
```

- Schlüsselwort `static` innerhalb Methoden/Funktionen

```
1  void machwas {
2    static int i = 0;
3    std::cout << i++ << std::endl;
4  }
```

- Schlüsselwort `static` in Source-Dateien

```
1  static int i; // kann *nur* in dieser Source-Datei
2                // verwendet werden
3  void my_func() {
4    std::cout << i++ << std::endl;
5  }
```

Outline

1. Mikrocontrollerprogrammierung
   - Alignment-Probleme
   - Interrupt-Kontext
2. Effizientes C++
   - Virtuelle Methoden
   - Das Schlüsselwort `inline`
   - Auch Kleinvieh macht zum Teil viel Mist
3. Kleinere Fallstricke...
4. Weiterführende Literatur

Weiterführende Literatur

- C++ in Embedded Systems: Myth and Reality
  `http://www.embedded.com/98/9802fe3.htm`
- Technical Report on C++ Performance
  `http://www.research.att.com/~bs/performanceTR.pdf`