Optimierung durch Ganzzahl-Arithmetik für Mikrokontroller

Martin Ongsiek

29.10.2011 / LABORTAGE

Gliederung

Vorsicht vor Optimierungen!

Rules of Optimization:
Rule 1: Don't do it.
Rule 2 (for experts only): Don't do it yet.
Michael A. Jackson

- Zeitverschwendung
- Wartung und Verständis ist schwerer
- Fehler
- Notwendig?

http://www.clean-code-developer.de/Vorsicht-vor-Optimierungen.ashx

Vorsicht vor Optimierungen!

Rules of Optimization:
Rule 1: Don't do it.
Rule 2 (for experts only): Don't do it yet.
Michael A. Jackson

- Zeitverschwendung
- Wartung und Verständis ist schwerer
- Fehler
- Notwendig?

http://www.clean-code-developer.de/Vorsicht-vor-Optimierungen.ashx

Vorsicht vor Optimierungen!

Rules of Optimization:
Rule 1: Don't do it.
Rule 2 (for experts only): Don't do it yet.
Michael A. Jackson

- Zeitverschwendung
- Wartung und Verständis ist schwerer
- Fehler
- Notwendig?

http://www.clean-code-developer.de/Vorsicht-vor-Optimierungen.ashx

Vorsicht vor Optimierungen!

Rules of Optimization:
Rule 1: Don't do it.
Rule 2 (for experts only): Don't do it yet.
Michael A. Jackson

- Zeitverschwendung
- Wartung und Verständis ist schwerer
- Fehler
- Notwendig?

http://www.clean-code-developer.de/Vorsicht-vor-Optimierungen.ashx

Vorsicht vor Optimierungen!

Rules of Optimization:
Rule 1: Don't do it.
Rule 2 (for experts only): Don't do it yet.
Michael A. Jackson

- Zeitverschwendung
- Wartung und Verständis ist schwerer
- Fehler
- Notwendig?

http://www.clean-code-developer.de/Vorsicht-vor-Optimierungen.ashx

Wie sollte man vorgehen?

- Die Software so einfach halten wie möglich.
- Das Rad nicht neu erfinden.
  Bibithecksfunktionen sind bereits optimiert und getestet.
- Vermeidung von blockierenden Code. z.B. delay(100);
  Betriebssystem notwendig?
- Zeit messen.
- Kritische Programmteile finden

Wie sollte man vorgehen?

- Die Software so einfach halten wie möglich.
- Das Rad nicht neu erfinden.
  Bibithecksfunktionen sind bereits optimiert und getestet.
- Vermeidung von blockierenden Code. z.B. delay(100);
  Betriebssystem notwendig?
- Zeit messen.
- Kritische Programmteile finden

Wie sollte man vorgehen?

- Die Software so einfach halten wie möglich.
- Das Rad nicht neu erfinden.
  Bibithecksfunktionen sind bereits optimiert und getestet.
- Vermeidung von blockierenden Code. z.B. delay(100);
  Betriebssystem notwendig?
- Zeit messen.
- Kritische Programmteile finden

Wie sollte man vorgehen?

- Die Software so einfach halten wie möglich.
- Das Rad nicht neu erfinden.
  Bibithecksfunktionen sind bereits optimiert und getestet.
- Vermeidung von blockierenden Code. z.B. delay(100);
  Betriebssystem notwendig?
- Zeit messen.
- Kritische Programmteile finden

Wie sollte man vorgehen?

- Die Software so einfach halten wie möglich.
- Das Rad nicht neu erfinden.
  Bibithecksfunktionen sind bereits optimiert und getestet.
- Vermeidung von blockierenden Code. z.B. delay(100);
  Betriebssystem notwendig?
- Zeit messen.
- Kritische Programmteile finden

Zahlensysteme

C-Grundtyp	int-Type	Wertebeich
unsigned char	uint8_t	0 . . . 255 (0x00 . . . 0xff)
signed char	int8_t	-128 . . . 127 (0x80 . . . 0xff, 0 . . . 0x7f)
unsigned short	uint16_t	0 . . . 65535
signed short	int16_t	-32.768 . . . 32.767
unsigned long	uint32_t	0 . . . 4.294.967.295
signed long	int32_t	-2.147.483.648 . . . 2.147.483.647

unsigned algemein	für n Bits	$0 \ldots (2^n - 1)$
signed algemein	für n Bits	$-(2^{n-1}) \ldots (2^{n-1} - 1)$

Zeit messen

- Auf das Wesentliche beschränken.
- Störer abschalten, z.B. Interrupts.
- Einen Timer zum Zeit messen verwenden.

Zeit messen

- Auf das Wesentliche beschränken.
- Störer abschalten, z.B. Interrupts.
- Einen Timer zum Zeit messen verwenden.

Zeit messen

- Auf das Wesentliche beschränken.
- Störer abschalten, z.B. Interrupts.
- Einen Timer zum Zeit messen verwenden.

Zeit messen von Funktionen

```
#define TIME_FUNC(func) print_exe_time(func,""#func)
typedef void (*test_func_t)(void);
void print_exec_time(test_func_t func, char name[]) {
  TCCR1A = 0; TCCR1B = 0; TCNT1 = 0; TIFR  = 0;
  TCCR1B = 1; // prescaler 1 Timer starten
  func();
  TCCR1B = 0; // Timer stoppen
  PrintSignedShortFormated(TCNT1);
  PrintString(" : ");
  PrintString(name);
  PrintString("\n");
}
```

Zeitvergleichsmessungen mit mehreren AVR-Kompilern

avr-gcc 4.3.3 -Os	avr-gcc 4.3.3 -O2	avr-gcc 4.3.3 -O3	avr-gcc 4.6.1 -Os	IAR 5.51/6.1 speed	
573	573	573	583	243	float_convert_to_float
358	358	358	369	213	float_convert_to_int
932	932	932	938	257	float_add_const
849	849	849	857	196	float_add
1973	1973	1973	1960	234	float_mul_const
2039	2039	2039	2014	229	float_mul
1403	1403	1403	1355	772	float_div_const
1358	1358	1358	1310	748	float_div
221	221	221	221	222	int16_div10
204	204	204	206	48	int16_div10_fast
37	23	23	37	30	int16_div16
222	222	222	222	227	int16_div
30	30	30	29	27	int16_mult_const
32	32	32	31	33	int16_mult
614	614	614	614	597	int32_div10
53	53	53	53	72	int32_div16
633	633	633	633	630	int32_div
77	77	77	79	49	int32_mult_const
498	498	498	500	2661	float_sqrt
356	356	15	356	422	int16_sqrt
204	204	204	204	---	int16_sqrt_asm
1468	1340	260	1459	1638	int32_sqrt
568	568	568	568	---	int32_sqrt_asm
5519	5519	5519	5446	3294	float_sin
117	117	117	114	101	int16_sin
1028	1010	480	1395	876 / 786	test_loop1
910	910	910	711	812 / 713	test_loop2
917	910	910	716	814 / 715	test_loop3
480	480	480	461	452 / 443	test_loop4
186	153	153	205	205	int16_pi_regler
9	9	1	9	10	empty_func
1285	1234	104	1187	320	print_int16
1283	1280	1280	1273	---	print_int16_itoa
8362	8390	18190	8394	2509	Größe in Bytes

Beispiel unoptimiert

```
for (uint8_t t = 0;
t < 100; t++) {
  for (uint8_t x = 0; x < 16; x++) {
    for (uint8_t y = 0; y < 16; y++) {
      buffer[x][y] = sqrt(t*t*x*x)*sin(y*y) + 566*t*y/300
                     + 3*t*t;
    }
  }
  ausgabe(buffer);
}
```

Optimierung Schritt 1

Berrechnungen nicht unnötig wiederholen.

```
for (uint8_t t = 0; t < 100; t++) {
  uint32_t h1 = 566L*t, h2 = t*t; // 256 mal seltener
  uint32_t h3 = h2*3;
  for (uint8_t x = 0; x < 16; x++) {
    uint16_t h4 = t*x; // 16 mal seltene sqrt unnötig
    for (uint8_t y = 0; y < 16; y++) {
      buffer[x][y] = h4*sin(y*y) + h1*y/300 + h3;
    }
  }
  ausgabe(buffer);
}
```

Optimierung Schritt 2

Division ist langsam!
Es sei denn, es wird durch eine 2er Potenzen geteilt.

$$\frac{\overbrace{566}^{k} \cdot t \cdot y}{300} = \frac{k \cdot c \cdot t \cdot y}{\underbrace{300 \cdot c}_{2^n}} \approx \frac{483 \cdot t}{256}$$

$$n = 8 \Rightarrow c = \frac{256}{300} \Rightarrow k \cdot c = \frac{256 \cdot 566}{300} = 482,9866$$

$$Fehler = \frac{482,9866 - 483}{482,9866} = 0,00276\%$$

Optimierung Schritt 2

Division ist langsam!
Es sei denn, es wird durch eine 2er Potenzen geteilt.

$$\frac{\overbrace{566}^{k} \cdot t \cdot y}{300} = \frac{k \cdot c \cdot t \cdot y}{\underbrace{300 \cdot c}_{2^n}} \approx \frac{483 \cdot t}{256}$$

$$n = 8 \Rightarrow c = \frac{256}{300} \Rightarrow k \cdot c = \frac{256 \cdot 566}{300} = 482,9866$$

$$Fehler = \frac{482,9866 - 483}{482,9866} = 0,00276\%$$

Optimierung Schritt 2

Division ist langsam!
Es sei denn, es wird durch eine 2er Potenzen geteilt.

$$\frac{\overbrace{566}^{k} \cdot t \cdot y}{300} = \frac{k \cdot c \cdot t \cdot y}{\underbrace{300 \cdot c}_{2^n}} \approx \frac{483 \cdot t}{256}$$

$$n = 8 \Rightarrow c = \frac{256}{300} \Rightarrow k \cdot c = \frac{256 \cdot 566}{300} = 482,9866$$

$$Fehler = \frac{482,9866 - 483}{482,9866} = 0,00276\%$$

Optimierung Schritt 2

Division ist langsam!
Es sei denn, es wird durch eine 2er Potenzen geteilt.

$$\frac{\overbrace{566}^{k} \cdot t \cdot y}{300} = \frac{k \cdot c \cdot t \cdot y}{\underbrace{300 \cdot c}_{2^n}} \approx \frac{483 \cdot t}{256}$$

$$n = 8 \Rightarrow c = \frac{256}{300} \Rightarrow k \cdot c = \frac{256 \cdot 566}{300} = 482,9866$$

$$Fehler = \frac{482,9866 - 483}{482,9866} = 0,00276\%$$

Optimierung Schritt 2

Division ist langsam!
Es sei denn, es wird durch eine 2er Potenzen geteilt.

$$\frac{\overbrace{566}^{k} \cdot t \cdot y}{300} = \frac{k \cdot c \cdot t \cdot y}{\underbrace{300 \cdot c}_{2^n}} \approx \frac{483 \cdot t}{256}$$

$$n = 8 \Rightarrow c = \frac{256}{300} \Rightarrow k \cdot c = \frac{256 \cdot 566}{300} = 482,9866$$

$$Fehler = \frac{482,9866 - 483}{482,9866} = 0,00276\%$$

Optimierung Schritt 2

Division ist langsam!
Es sei denn, es wird durch eine 2er Potenzen geteilt.

$$\frac{\overbrace{566}^{k} \cdot t \cdot y}{300} = \frac{k \cdot c \cdot t \cdot y}{\underbrace{300 \cdot c}_{2^n}} \approx \frac{483 \cdot t}{256}$$

$$n = 8 \Rightarrow c = \frac{256}{300} \Rightarrow k \cdot c = \frac{256 \cdot 566}{300} = 482,9866$$

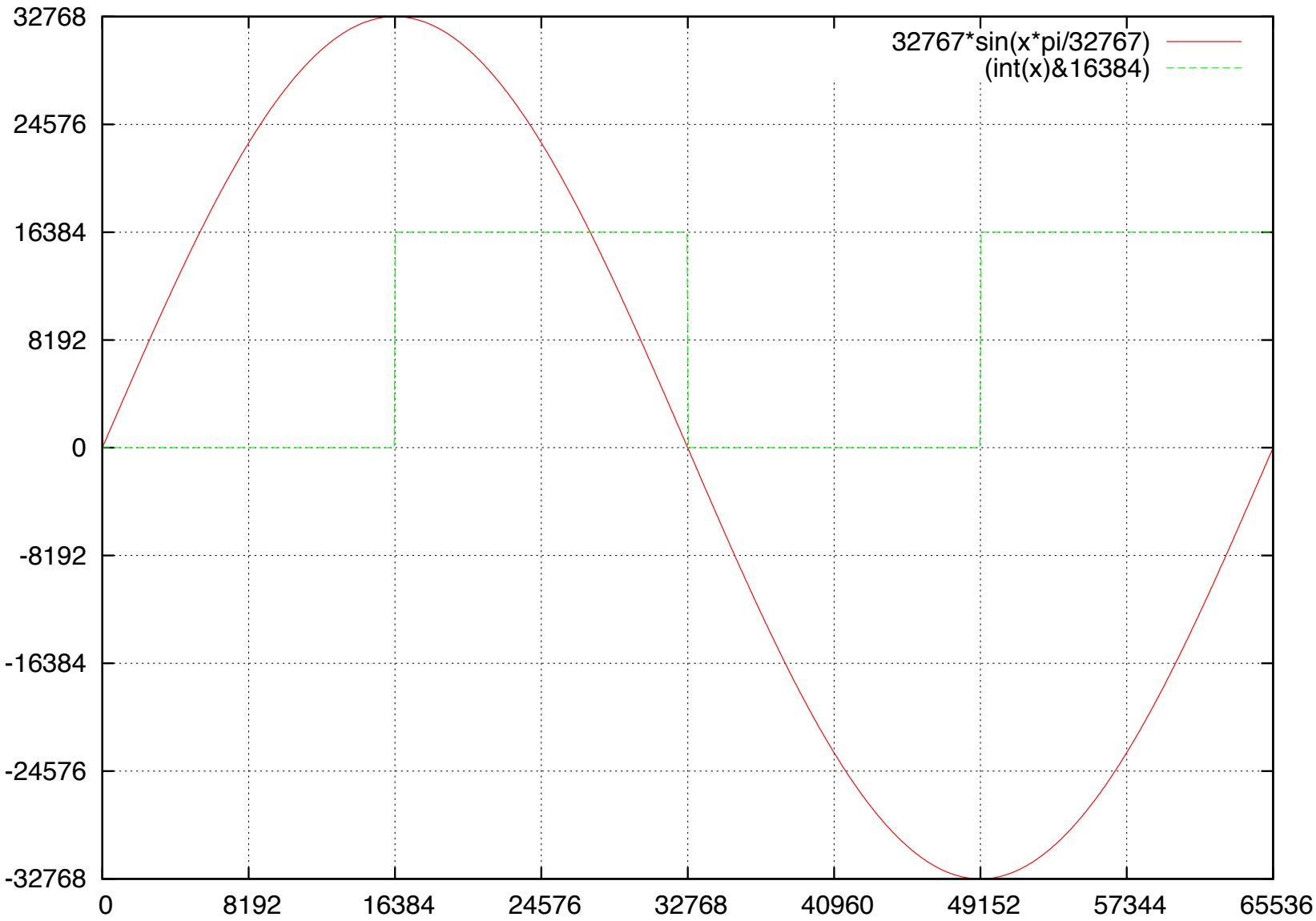
$$Fehler = \frac{482,9866 - 483}{482,9866} = 0,00276\%$$

Optimierung Schritt 3

Berrechnungen nicht unnötig wiederholen. Es werden nur 16 Sinuswerte benötigt, diese kann man in einer Tabelle ablegen.

```
for (uint8_t t = 0; t < 100; t++) {
  uint32_t h1 = ((256*566+150)/300)*t, h2 = t*t;
  uint32_t h3 = h2*3;
  for (uint8_t x = 0; x < 16; x++) {
    uint16_t h4 = t*x;
    for (uint8_t y = 0; y < 16; y++) {
      buffer[x][y] = h4*sin_tab[y] + ((h1*y) >> 8) + h3;
    }
  }
  ausgabe(buffer);
}
```

Lookuptable mit Interpolation am Beispiel Sinus 1



Lookuptable mit Interpolation am Beispiel Sinus 2

```
int16_t PROGMEM sin_table[66] = {0, 804, 1608 ... 32757};
static int16_t Sine(uint16_t phase) {
    int16_t s0;
    uint16_t tmp_phase= phase & 0x7fff, tmp_phase_hi;
    if (tmp_phase & 0x4000) //90 bis 180° 240 bis 360°
        tmp_phase = 0x8000 - tmp_phase; //Tab. rückwärts
    tmp_phase_hi = tmp_phase >> 8; // 0...64
    s0 = PW(sin_table[tmp_phase_hi]); // Star
    s0 += ((int16_t)((((int32_t)(PW(sin_table[
        tmp_phase_hi+1]) - s0))*(tmp_phase&0xff))>>8));
    if (phase & 0x8000)  s0 = -s0; // Sinus ab 180° negativ
    return s0;
}
```

Lookuptable mit Interpolation am Beispiel Sinus 3

float sin in	float sin out	int sin in	int sin out
$0,0$	$0,0$	0	0
$\frac{\pi}{2}$	$1,0$	0x4000	0x7FFF
$\frac{3\cdot\pi}{2}$	$-1,0$	0xC000	0x8001

Floatingpoint: sin(x)
Integer: Sine(x_int)
Umwandlung von x in Bogenmaß nach x_int $\frac{x\cdot 0x8000}{\pi} = 10430,06$
Umwandlung von x in Grad nach x_int $\frac{x\cdot 0x8000}{180^\circ} = 182$

Optimierung Schritt 4

```
for (uint8_t t = 0; t < 100; t++) {
  uint32_t h1 = ((256*566+150)/300)*t, h2 = t*t;
  uint32_t h3 = h2*3;
  for (uint8_t x = 0; x < 16; x++) {
    uint32_t h4 = isqrt32(h2*x*x);  // 16 mal seltener
    for (uint8_t y = 0; y < 16; y++) {
      buffer[x][y] = (h4*Sine(10430L*y*y)) >> 15) +
                     ((h1*y) >> 8) + h3;
    }
  }
  ausgabe(buffer);
}
```

Küstliche Nachkommastellen

Durch Multiplikation eines Faktors f zur einer Variablen x kann man sich künstliche Nachkommastellen aus Ganzzahlen machen. Am effizentesten sind dabei 2er Protenz, da Dividieren und und Modulo rechenaufwendig sind.

$$x_n = x \cdot f \Rightarrow x_n^2 = x \cdot f \cdot x \cdot \overbrace{f}^{zuviel}$$

$$\frac{2 \cdot x_n}{y_n} = \frac{2 \cdot x \cdot f}{y \cdot \underbrace{f}_{zuviel}}$$

Am Besten eine Einheit verwenden, in der man keine Nachkommastellen benötigt.

Küstliche Nachkommastellen

Durch Multiplikation eines Faktors f zur einer Variablen x kann man sich künstliche Nachkommastellen aus Ganzzahlen machen. Am effizentesten sind dabei 2er Protenz, da Dividieren und und Modulo rechenaufwendig sind.

$$x_n = x \cdot f \Rightarrow x_n^2 = x \cdot f \cdot x \cdot \overbrace{f}^{zuviel}$$

$$\frac{2 \cdot x_n}{y_n} = \frac{2 \cdot x \cdot f}{y \cdot \underbrace{f}_{zuviel}}$$

Am Besten eine Einheit verwenden, in der man keine Nachkommastellen benötigt.

Küstliche Nachkommastellen

Durch Multiplikation eines Faktors f zur einer Variablen x kann man sich künstliche Nachkommastellen aus Ganzzahlen machen. Am effizentesten sind dabei 2er Protenz, da Dividieren und und Modulo rechenaufwendig sind.

$$x_n = x \cdot f \Rightarrow x_n^2 = x \cdot f \cdot x \cdot \overbrace{f}^{zuviel}$$

$$\frac{2 \cdot x_n}{y_n} = \frac{2 \cdot x \cdot f}{y \cdot \underbrace{f}_{zuviel}}$$

Am Besten eine Einheit verwenden, in der man keine Nachkommastellen benötigt.

Küstliche Nachkommastellen

Durch Multiplikation eines Faktors f zur einer Variablen x kann man sich künstliche Nachkommastellen aus Ganzzahlen machen. Am effizentesten sind dabei 2er Protenz, da Dividieren und und Modulo rechenaufwendig sind.

$$x_n = x \cdot f \Rightarrow x_n^2 = x \cdot f \cdot x \cdot \overbrace{f}^{zuviel}$$

$$\frac{2 \cdot x_n}{y_n} = \frac{2 \cdot x \cdot f}{y \cdot \underbrace{f}_{zuviel}}$$

Am Besten eine Einheit verwenden, in der man keine Nachkommastellen benötigt.

Küstliche Nachkommastellen

Durch Multiplikation eines Faktors f zur einer Variablen x kann man sich künstliche Nachkommastellen aus Ganzzahlen machen. Am effizentesten sind dabei 2er Protenz, da Dividieren und und Modulo rechenaufwendig sind.

$$x_n = x \cdot f \Rightarrow x_n^2 = x \cdot f \cdot x \cdot \overbrace{f}^{zuviel}$$

$$\frac{2 \cdot x_n}{y_n} = \frac{2 \cdot x \cdot f}{y \cdot \underbrace{f}_{zuviel}}$$

Am Besten eine Einheit verwenden, in der man keine Nachkommastellen benötigt.