Echtzeitsysteme

Übungen zur Vorlesung

Entwicklungsumgebung (Teil 2)

Florian Schmaus Peter Wägemann

Friedrich-Alexander-Universität Erlangen-Nürnberg (FAU)
Lehrstuhl für Informatik 4 (Verteilte Systeme und Betriebssysteme)
https://www4.cs.fau.de

26.10.2017

Übersicht

1. Weiterführende Informationen zum Übungsbetrieb
2. Wahl geeigneter Datentypen
3. Handwerkszeug
   - Oszilloskop-Cursor
   - Tiefpassfilter
   - Pulsweitenmodulation
   - Zeit in eCos
4. Interruptbehandlung
   - ISR & DSR
   - eCos-Unterbrechungsbehandlung
5. Zusammenfassung

Informationen zum Übungsbetrieb

- Accounts im CIP-Pool?
  - `cipan`
  - Sprechstunde der CIP-Admins
- Kenntnisse im Umgang mit Terminals hilfreich
  - `UNIX` Vorkurs der FSI
  - `https://fsi.cs.fau.de/dw/informationen/ese/2016ws/vorkurs`
- Arbeiten zu Hause?
  - `https://gitlab.cs.fau.de/ezs/ezs-board/wikis`
  - `https://gitlab.cs.fau.de/ezs/stm32-blackmagic`
  - Konfigurationen anpassen
  - Idealerweise nativ unter Linux

Wiederholungen notwendig?

- Einführung in GDB
  - EZS-Dashboard
- Stack-Aufbau bei Funktionsaufrufen
  - Stack-Tiefe
  - Backtrace
- Komponenten der Toolchain
  - Präprozessor
  - Compiler
  - Flasher
  - ...

Bekannte Probleme & Beliebte Fehler

- Problem: Debugger reagiert nicht, Board lässt sich nicht flashen
  - ☞ USB-Kabel abstecken & anstecken
- Falls immer noch keine Reaktion
  - ☞ `make unbrick`, Anweisungen des Befehls folgen
- Keine leuchtenden LEDs
  - ☞ anderes Board probieren
- Boxen verwenden
- Vorsicht bei wackeligen Jumper-Kabeln
- `make gdb` weniger fragil als gdb-Dashboard (`make debug`)
- Vorsicht bei `printf`-Debugging
- Verwendung von Fließkomma-Arithmetik und Typecasts ...

Übersicht

Frage #1

Zu was wird `7/2` ausgewertet?

1. `3.5`
2. 3
3. nicht definiert in C

Erklärung

- Standard-Typ für Ganzzahlen ist **`int`**
- Rest verschwindet bei Ganzzahl-Division

Frage #2

Zu was wird `2/7` ausgewertet?

1. 1
2. `0`
3. nicht definiert in C

Erklärung

- Standard-Typ für Ganzzahlen ist **`int`**
- Rest verschwindet bei Ganzzahl-Division

Frage #3

Zu was wird `7/2.` ausgewertet?

1. `immer noch 3`
2. `0`
3. `3.5`

Erklärung

- `2.0 == 2.` $\rightsquigarrow$ **`double`** auf der rechten Seite
- `7` wird in diesem Ausdruck als **`double`** behandelt, auch linke Seite
- Division zweier **`double`** Werte

Frage #5

Zu was wird `1/2 + 1/2` ausgewertet?

1. nicht definiert
2. `0`
3. `1` (dank Compileroptimierung)

Erklärung

- **`int`**$_1$/<größerer **`int`**$_2$> $\rightsquigarrow$ `0 + 0 = 0`
- Compileroptimierung nicht C-Konform

Frage #6

Zu was wird `2 * M_PI` ausgewertet?

1. `6`
2. ungefähr `6.28`
3. `6.28318530717958647692528676655900576839433879875 0...`

Erklärung

- `M_PI` $\rightsquigarrow$ **`double`**
- **`double`** Standard-Typ, außer zusätzliches Literal (`3.14f`)
- Begrenzter Wertebereich:
  `6.28318530717958600000000000000000`

Frage #7

```
1  double a = 0.1;
2  double b = 0.2;

3  float aa = 0.1;
4  float bb = 0.2;

5  if (a+b == aa+bb){
6    ezs_printf("equal\n");
7  }else{
8    ezs_printf("unequal: %.30f != %.30f\n", (a+b), (aa+bb));
9  }
```

Was wird ausgegeben?

1. equal
2. unequal...

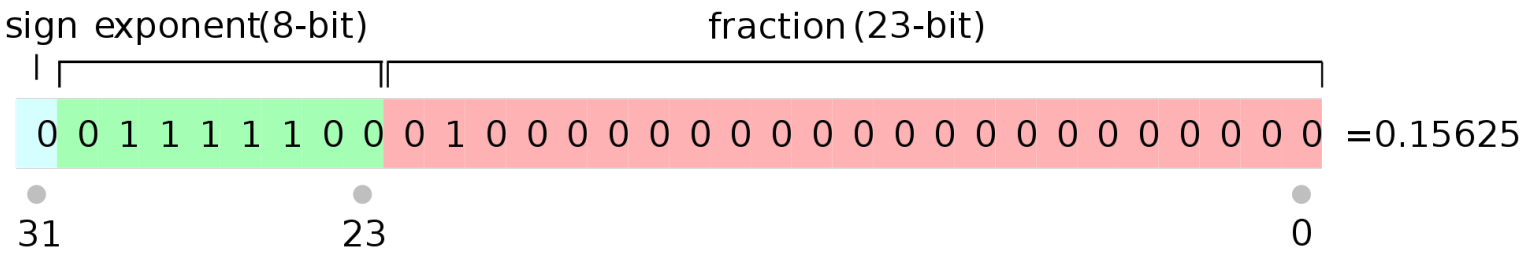
Fließkomma-Arithmetik

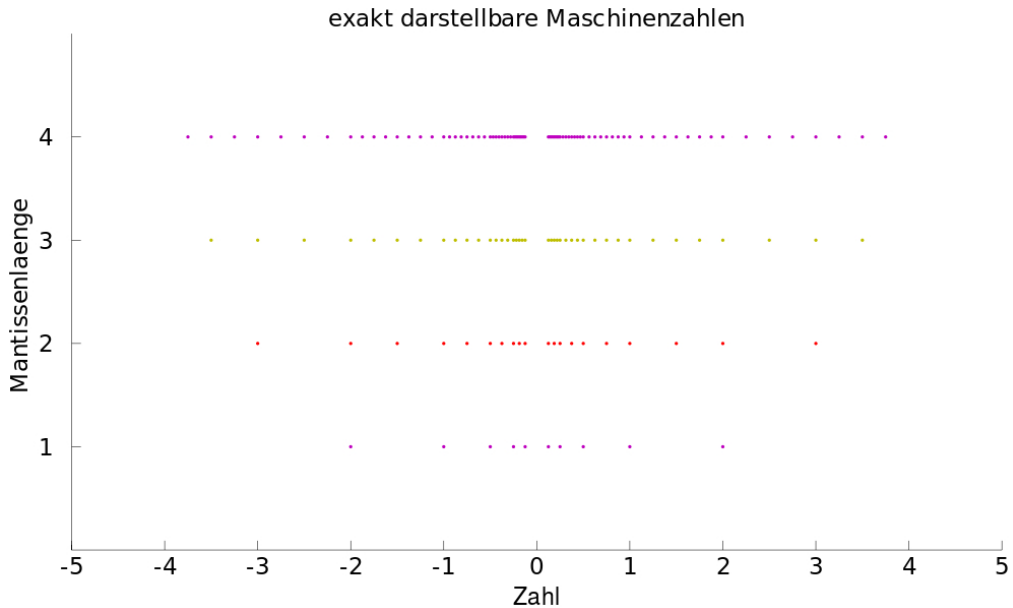
```
1   double a = 0.1;
2   double b = 0.2;

3   float aa = 0.1;
4   float bb = 0.2;

5   if (a+b == aa+bb){
6     ezs_printf("equal\n");
7   }else{
8     ezs_printf("unequal: %.30f != %.30f\n", (a+b), (aa+bb));
9   }
```

```
1  /* Ausgabe:
2  unequal:
3  0.300000000000000000000000000000 !=
4  0.300000011920929000000000000000
5  */
```

- Angenommen die Einheit ist Sekunden
  - 11,9 ns Fehler durch *einzelne Berechnung*
  - Kumulation der Rundungsfehler

Begrenzte Wertebereiche – IEEE 754





- IEEE 754
  - `sizeof(float) == 4`
  - `sizeof(double) == 8`

Probleme begrenzter Wertebereiche

- *What Every Computer Scientist Should Know About Floating-Point Arithmetic* [1]
- Rundungsfehler & Überläufe äußerst kritisch in *harten Echtzeitsystemen*
- Konvertierungen zwischen Größeneinheiten (`sec_to_nanosec: * 1e9`)
- Vermeidung des Wechsels von Größeneinheiten
- Verwendung von Festkomma-Arithmetik $\rightsquigarrow$ VEZS '17
- Integer-Division ist *kein sicherer Ausweg*

☞ *Sorgfalt bei arithmetischen Operationen in begrenzten Wertebereichen*

Wahl des Datentyps bei Berechnung des Sinus-Wertes

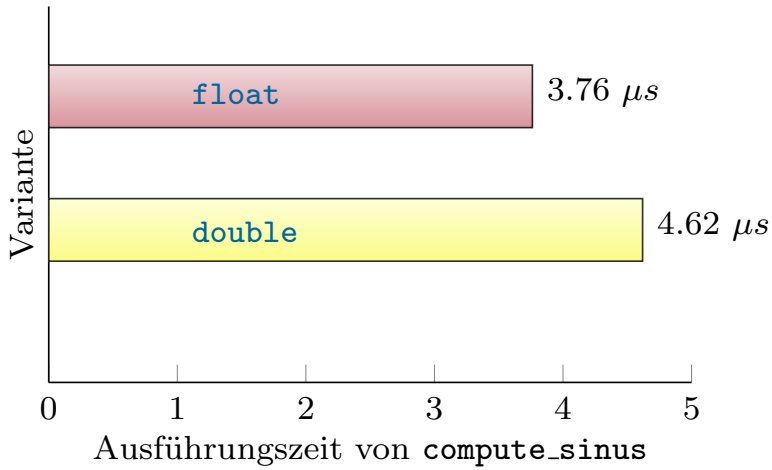
- Harmonische Schwingung[1]: $y(t) = y_0 \cdot sin(\omega t + \varphi_0)$ und $\omega = 2\pi f$

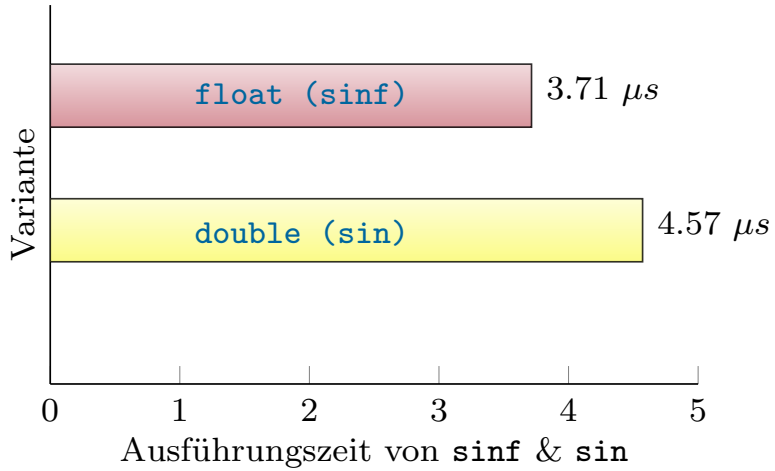
```
1  #define TYPE {int|double|float} ?
2  ...
3  TYPE compute_sinus(OTHER_TYPE real_time) {
4    TYPE f     = ...
5    TYPE omega = 2 * M_PI * f;
6    ...
7      ... sin(omega * real_time) // oder doch sinf(omega * real_time) ?
8    ...
9  }
```

- **`float`** *oder* **`double`** *für Realzeit sinnvoll? Was ist* `OTHER_TYPE`*?*
- Konfiguration von **`float`** und **`double`** sinnvoll
- *Laufzeit von* `compute_sinus()`*?*

[1] https://de.wikipedia.org/wiki/Schwingung#Harmonische_Schwingung

Vergleich der Laufzeiten

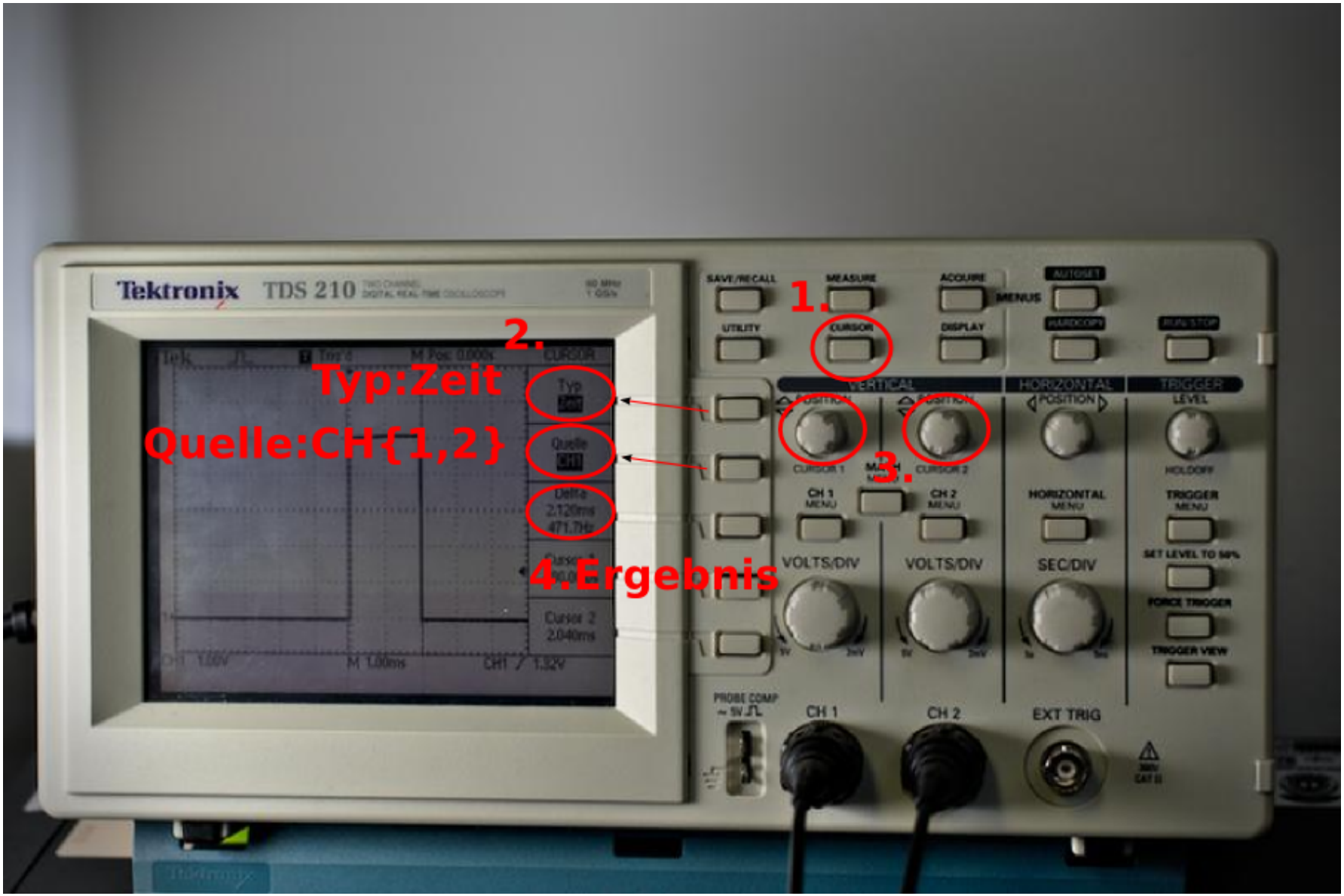




- Laufzeitzuwachs um *23 %* bei Wechsel **`float`** → **`double`**
- Soft Float? Hard Float? hier: Soft Float
- Noch mehr Optimierungspotential? Wo wird die Laufzeit verbraucht?
  - *99 %* der Gesamtlaufzeit für `sinf` und `sin`
- Wahl des Datentyps in Abhängigkeit der Wortbreite (32-Bit Cortex-M4, 8-Bit AVR)
- Spezialbibliothek für Signalverarbeitung mit Integer-Arithmetik
- Spezielle Hardware-Einheiten zur Signalverarbeitung

Übersicht

Cursor


Tektronix TDS 210
1.
2.
Typ:Zeit
Quelle:CH{1,2}
3.
4.Ergebnis
SAVE/RECALL
MEASURE
ACQUIRE
UTILITY
CURSOR
DISPLAY
VERTICAL
HORIZONTAL
TRIGGER
VOLTS/DIV
VOLTS/DIV
SEC/DIV
CH 1
CH 2
EXT TRIG

Tiefpassfilter

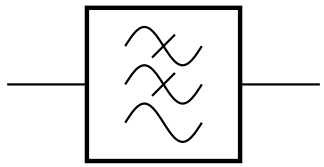


Abbildung: Schaltbild RC-Glied

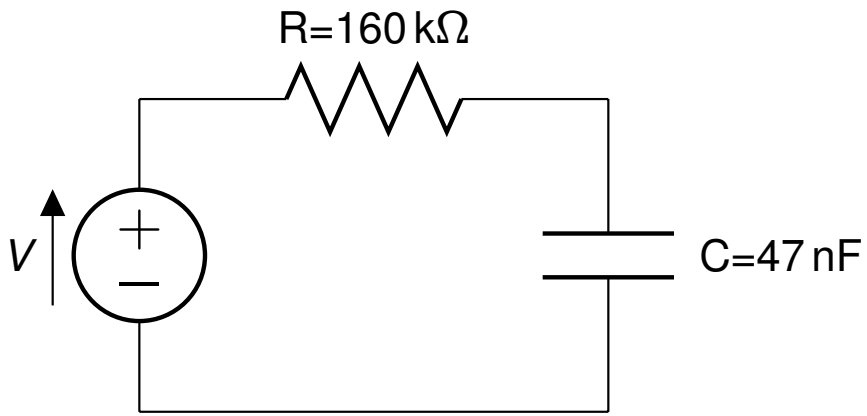
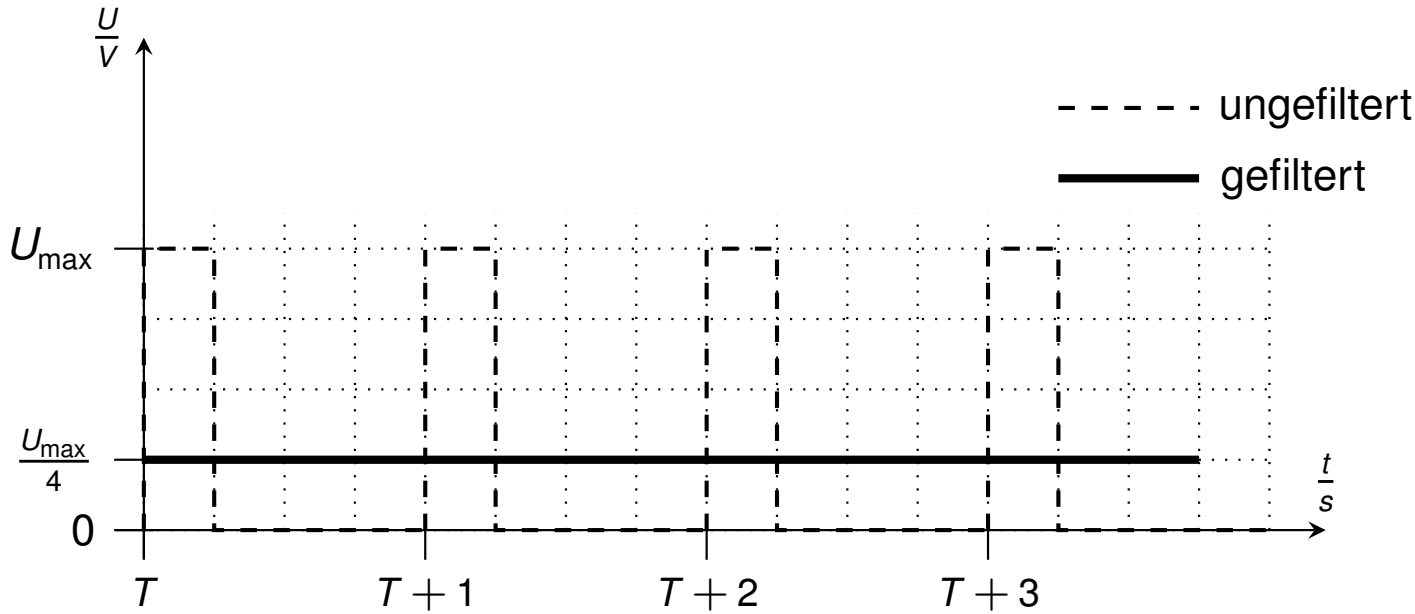


Abbildung: Schaltung RC-Filter auf EZS-Board

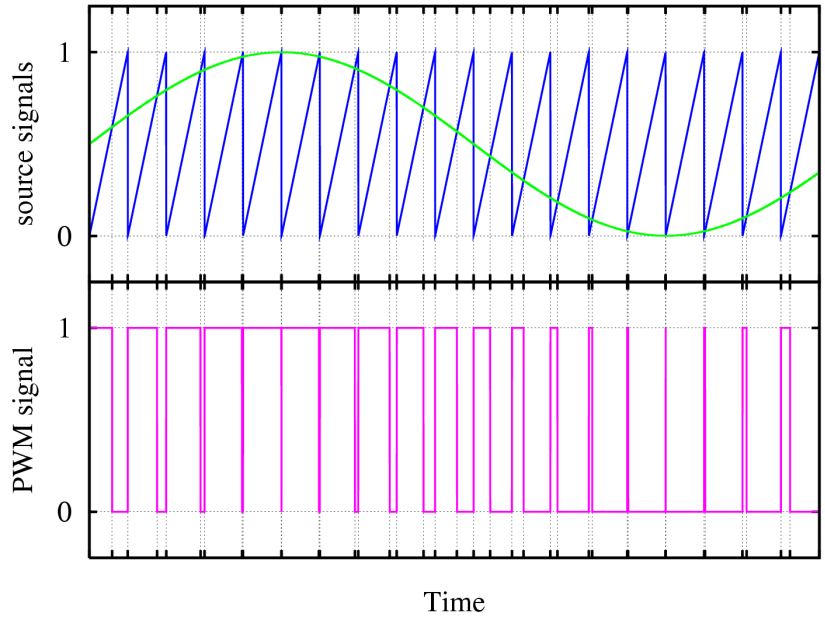
- Filterung hochfrequenter Schwingungen
- Zeitkonstante $\tau = R \cdot C = 7,52\text{ms}$
- *Grenzfrequenz* (Dämpfung um 3dB $\approx 71\,\%$): $f_c = \frac{1}{2 \cdot \pi \cdot \tau} = 21\,Hz$

Digital-Analog Umwandlung & Pulsweitenmodulation I



- Einschaltdauer proportional zum Mittelwert des Ausgangssignals
- Variation der Pulsweite oder Einschaltdauer (engl. duty cycle)
- Pulsweitenmodulation (engl. pulse-width modulation, PWM)
- „Pseudo“ Digital-Analog-Wandler (engl. digital-analog converter, DAC)

Digital-Analog Umwandlung & Pulsweitenmodulation II



Verfahren zur **Signalerzeugung**

- Hardware: Vergleich periodischer Zähler und Wert der Einschaltdauer
- Weit verbreitet: Motorsteuerung, Class-D-Verstärker, Schaltnetzteile, Nachrichtenübertragung,...
- Mittels Tiefpass $\rightsquigarrow$ Digital-Analog-Wandlung
- libEZS: `void ezs_dac_write(uint8_t)` (zusätzlich *echter* DAC auf Board)

Umgang mit Zeit in eCos

- Aktuelle Aufgabe: Ausführung soll um feste Zeit *verzögert* werden
  $\rightsquigarrow$ `cyg_thread_delay()` (bei uns `ezs_delay_us()`)
- Erwartet Parameter der Einheit *Clock-Ticks* – *Wieso?*
- Zeitmessung nur per Timer möglich $\rightsquigarrow$ Timer-Zyklus kleinste Einheit

`cyg_clock_get_resolution(cyg_real_time_clock())` liefert Auflösung der Echtzeituhr:

```
1  typedef struct {
2    cyg_uint32 dividend;
3    cyg_uint32 divisor;
4  } cyg_resolution_t;
```

- $\frac{\texttt{dividend}}{\texttt{divisor}}$ $\rightsquigarrow$ Zeit in ns, die ein Tick dauert (beispielsweise 1000 ns)
- *Warum Aufteilung in Dividend & Divisor?*
- Umrechnung sollte *einmalig* erfolgen

Übersicht

Wie behandle ich einen Interrupt?

Interrupt-Service-Routinen-Ausführung

- Unverzüglich, *asynchron*
  $\rightsquigarrow$ auch innerhalb von Kernelfunktionen!
- Innerhalb ISR *keine Systemaufrufe* erlaubt!
  $\Rightarrow$ Anmelden einer Deferrable Service Routine (DSR)

Deferrable-Service-Routinen-Ausführung

- *Synchron* zum Scheduler
- Falls Scheduler nicht verriegelt: *Unverzüglich* nach ISR
- sonst: Beim *Verlassen* des Kerns

Synonym: *Prolog-Epilog-Schema* bzw. *top/bottom half*

Wie behandle ich einen Interrupt?

Anmeldung von ISR und DSR[2]

```
#include <cyg/kernel/kapi.h>
void cyg_interrupt_create
(
 cyg_vector_t vector,
 cyg_priority_t priority,
 cyg_addrword_t data,
 cyg_ISR_t* isr,
 cyg_DSR_t* dsr,
 cyg_handle_t* handle,
 cyg_interrupt* intr
);
```

- Interruptvektornummer
- ⇝ Hardwarehandbuch

[2] http://ecos.sourceware.org/docs-latest/ref/kernel-interrupts.html

Wie behandle ich einen Interrupt?

Anmeldung von ISR und DSR[2]

```
#include <cyg/kernel/kapi.h>
void cyg_interrupt_create
(
 cyg_vector_t vector,
 cyg_priority_t priority,
 cyg_addrword_t data,
 cyg_ISR_t* isr,
 cyg_DSR_t* dsr,
 cyg_handle_t* handle,
 cyg_interrupt* intr
);
```

- Interruptpriorität
- für unterbrechbare Unterbrechungen (hardwareabhängig)

[2] http://ecos.sourceware.org/docs-latest/ref/kernel-interrupts.html

Wie behandle ich einen Interrupt?

Anmeldung von ISR und DSR[2]

```
#include <cyg/kernel/kapi.h>
void cyg_interrupt_create
(
 cyg_vector_t vector,
 cyg_priority_t priority,
 cyg_addrword_t data,
 cyg_ISR_t* isr,
 cyg_DSR_t* dsr,
 cyg_handle_t* handle,
 cyg_interrupt* intr
);
```

- Beliebiger Übergabeparameter für ISR/DSR

[2] http://ecos.sourceware.org/docs-latest/ref/kernel-interrupts.html

Wie behandle ich einen Interrupt?

Anmeldung von ISR und DSR[2]

```
#include <cyg/kernel/kapi.h>
void cyg_interrupt_create
(
 cyg_vector_t vector,
 cyg_priority_t priority,
 cyg_addrword_t data,
 cyg_ISR_t* isr,
 cyg_DSR_t* dsr,
 cyg_handle_t* handle,
 cyg_interrupt* intr
);
```

- Funktionszeiger auf *ISR-Implementierung*

Signatur:

```
cyg_uint32 (*)(cyg_vector_t, cyg_addrword_t)
```

[2] http://ecos.sourceware.org/docs-latest/ref/kernel-interrupts.html

Wie behandle ich einen Interrupt?

Anmeldung von ISR und DSR[2]

```
#include <cyg/kernel/kapi.h>
void cyg_interrupt_create
(
 cyg_vector_t vector,
 cyg_priority_t priority,
 cyg_addrword_t data,
 cyg_ISR_t* isr,
 cyg_DSR_t* dsr,
 cyg_handle_t* handle,
 cyg_interrupt* intr
);
```

- Funktionszeiger auf *DSR-Implementierung*

Signatur:

```
cyg_uint32 (*)(cyg_vector_t, cyg_ucount32, cyg_addrword_t)
```

[2] http://ecos.sourceware.org/docs-latest/ref/kernel-interrupts.html

Wie behandle ich einen Interrupt?

Anmeldung von ISR und DSR[2]

```
#include <cyg/kernel/kapi.h>
void cyg_interrupt_create
(
 cyg_vector_t vector,
 cyg_priority_t priority,
 cyg_addrword_t data,
 cyg_ISR_t* isr,
 cyg_DSR_t* dsr,
 cyg_handle_t* handle,
 cyg_interrupt* intr
);
```

- Handle und Speicher für *Interruptobjekt*

[2] http://ecos.sourceware.org/docs-latest/ref/kernel-interrupts.html

ISR-Implementierungsskelett

Beispiel einer minimalen ISR

```
cyg_uint32 isr(cyg_vector_t vector, cyg_addrword_t data) {
    cyg_bool_t dsr_required = 0;
    ...
    cyg_acknowledge_isr(vector);
    if (dsr_required) {
      return CYG_ISR_CALL_DSR;
    } else {
      return CYG_ISR_HANDLED;
    }
}
```

1. Beliebiger ISR-Code
2. Bestätigung der Interruptbehandlung

   Wozu ist das gut?
3. Anforderung einer DSR

   oder
4. Rückkehr ohne DSR

ISR-Implementierungsskelett

Beispiel einer minimalen ISR

```
cyg_uint32 isr(cyg_vector_t vector, cyg_addrword_t data) {
    cyg_bool_t dsr_required = 0;
    ...
    cyg_acknowledge_isr(vector);
    if (dsr_required) {
      return CYG_ISR_CALL_DSR;
    } else {
      return CYG_ISR_HANDLED;
    }
}
```

1. Beliebiger ISR-Code
2. Bestätigung der Interruptbehandlung

   Wozu ist das gut?
3. Anforderung einer DSR

   oder
4. Rückkehr ohne DSR

ISR-Implementierungsskelett

Beispiel einer minimalen ISR

```
cyg_uint32 isr(cyg_vector_t vector, cyg_addrword_t data) {
    cyg_bool_t dsr_required = 0;
    ...
    cyg_acknowledge_isr(vector);
    if (dsr_required) {
      return CYG_ISR_CALL_DSR;
    } else {
      return CYG_ISR_HANDLED;
    }
}
```

1. Beliebiger ISR-Code
2. Bestätigung der Interruptbehandlung

   Wozu ist das gut?
3. Anforderung einer DSR

   oder
4. Rückkehr ohne DSR

ISR-Implementierungsskelett

Beispiel einer minimalen ISR

```
cyg_uint32 isr(cyg_vector_t vector, cyg_addrword_t data) {
    cyg_bool_t dsr_required = 0;
    ...
    cyg_acknowledge_isr(vector);
    if (dsr_required) {
      return CYG_ISR_CALL_DSR;
    } else {
      return CYG_ISR_HANDLED;
    }
}
```

1. Beliebiger ISR-Code
2. Bestätigung der Interruptbehandlung

   Wozu ist das gut?
3. Anforderung einer DSR

   oder
4. Rückkehr ohne DSR

DSR-Implementierungskelett

Beispiel einer minimalen DSR

```
void dsr_function(
  cyg_vector_t vector,
  cyg_ucount32 count,
  cyg_addrword_t data)
{
...
}
```

1. Anzahl der ISRs, die diese DSR anforderten
   $\rightsquigarrow$ normalerweise 1
2. Ausführung *synchron* zum Scheduler
   Was bedeutet das?

DSR-Implementierungskelett

Beispiel einer minimalen DSR

```
void dsr_function(
  cyg_vector_t vector,
  cyg_ucount32 count,
  cyg_addrword_t data)
{
...
}
```

1. Anzahl der ISRs, die diese DSR anforderten

   $\rightsquigarrow$ normalerweise 1

2. Ausführung *synchron* zum Scheduler

   Was bedeutet das?

Übersicht

Zusammenfassung

- Vorsicht bei der **Wahl des Datentypen**
  - Abhängig von Zielarchitektur
  - Datentypen **beeinflussen die Laufzeit**
  - Geschwindigkeit *nicht notwendig, aber hilfreich* für Echtzeitsysteme
- Handwerkszeug für EZS
  - Arbeiten mit Oszilloskopen
  - DAC, PWM, Filter
  - Zeiten in eCos
- Interruptbehandlungen
  - Möglichst kurze ISR
  - Synchrone Abarbeitungen in DSR

Literatur

[1] David Goldberg.
What every computer scientist should know about floating-point arithmetic.
ACM Computing Surveys (CSUR), 23(1):5–48, 1991.