

Echtzeitsysteme

Übungen zur Vorlesung

Entwicklungsumgebung

Simon Schuster **Peter Wägemann**

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

Sommersemester 2022



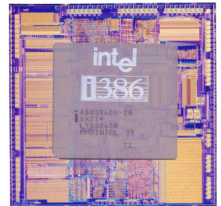
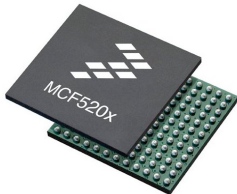
- 1 Einführung in eCos
- 2 Entwicklungsumgebung
 - Werkzeugkette und CMake
 - Blackmagic
 - Pulsweitenmodulation
 - Tiefpassfilter
 - Oszilloskop-Bedienung
- 3 Debuggen mit GDB



- 1 Einführung in eCos
- 2 Entwicklungsumgebung
 - Werkzeugkette und CMake
 - Blackmagic
 - Pulsweitenmodulation
 - Tiefpassfilter
 - Oszilloskop-Bedienung
- 3 Debuggen mit GDB



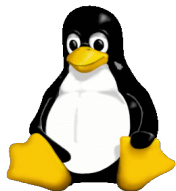
Prozessorvielfalt in der Echtzeitwelt



Noch mehr Betriebssysteme

free **RTOS**

μ Clinux



eCos

QNX

TinyOS


μ C/OS-II™
The Real-Time Kernel

HIGH TEC



eCos is an embedded, highly configurable, open-source, royalty-free, real-time operating system.

- Ursprünglich von der Fa. Cygnus Solutions entwickelt (1997)
- Primäres Entwurfsziel:
 - „deeply embedded systems“
 - „high-volume application“
 - „consumer electronics, telecommunications, automotive, . . .“
- Zusammenarbeit mit Redhat (1999)
- Seit 2002 quelloffen (GPL)

 <http://ecos.sourceforge.org>

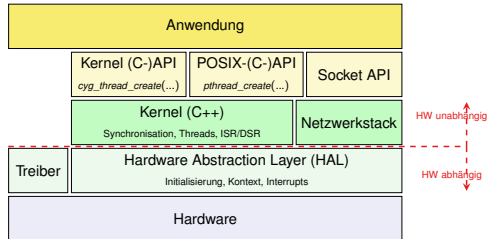


Unterstützte Plattformen

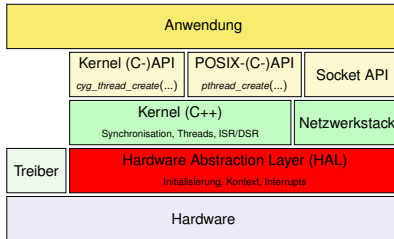
<http://www.ecoscentric.com/ecos/examples.shtml>

- Fujitsu SPARClite
- Matsushita MN10300
- Motorola PowerPC
- Advanced RISC Machines (ARM)
- Toshiba TX39
- Infineon TriCore
- Hitachi SH3
- NEC VR4300
- MB8683X
- ☞ *ARM Cortex*
- ☞ *Intel x86*
- ...

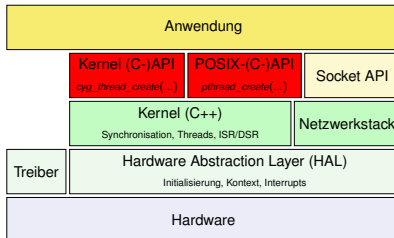




- Abstrahiert CPU- und plattformspezifische Eigenschaften
 - Kontextwechsel
 - Interruptverwaltung
 - CPU-Erkennung, Startup
 - Zeitgeber, I/O-Registerzugriffe

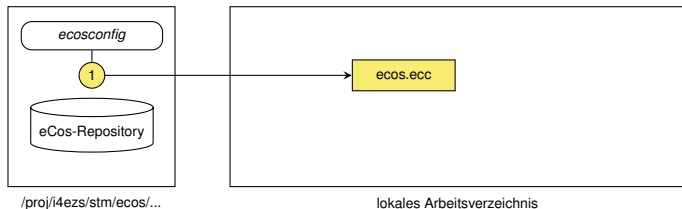


- Kernel API
 - vollständige C-Schnittstelle
 - siehe Dokumentation¹
- (Optionale) POSIX-Kompatibilitätsschicht
 - Scheduling-Konfiguration, *pthread_**
 - Timer, Semaphore, Message Queues, Signale, ...

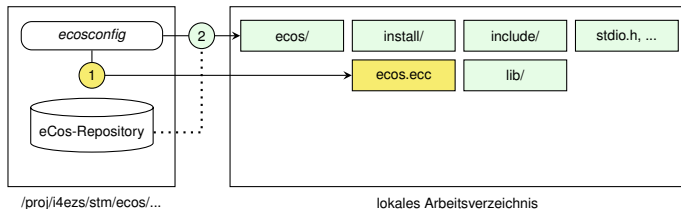


¹<http://ecos.sourceforge.org/docs-2.0/ref/ecos-ref.html>

1 Erstellen einer Konfiguration (configtool/ecosconfig)

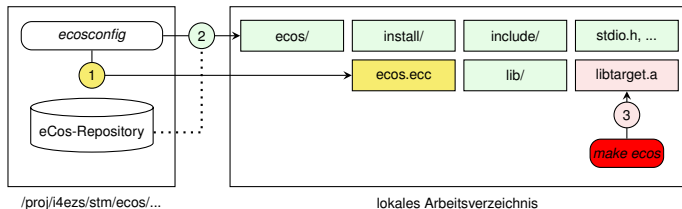


- 1 Erstellen einer Konfiguration (configtool/ecosconfig)
- 2 Kopieren ausgewählter Komponenten (configtool/ecosconfig)



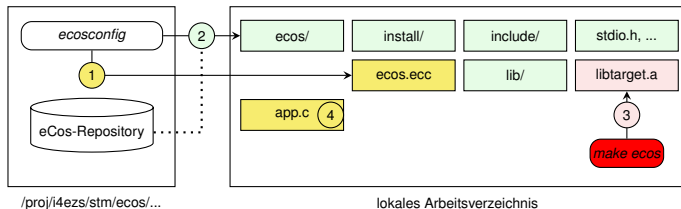
eCos-Entwicklungszyklus

- 1 Erstellen einer Konfiguration (configtool/ecosconfig)
- 2 Kopieren ausgewählter Komponenten (configtool/ecosconfig)
- 3 Erstellen einer Betriebssystembibliothek



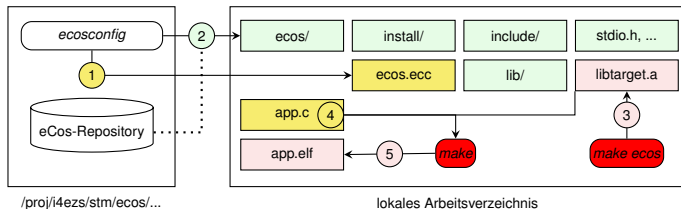
eCos-Entwicklungszyklus

- 1 Erstellen einer Konfiguration (configtool/ecosconfig)
- 2 Kopieren ausgewählter Komponenten (configtool/ecosconfig)
- 3 Erstellen einer Betriebssystembibliothek
- 4 Entwicklung der eigentlichen Anwendung



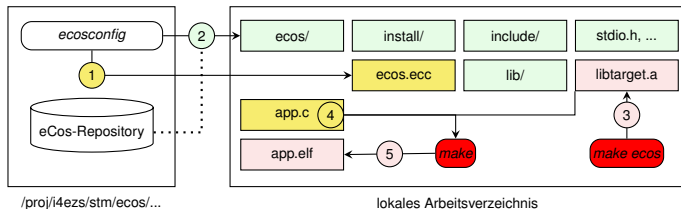
eCos-Entwicklungszyklus

- 1 Erstellen einer Konfiguration (configtool/ecosconfig)
- 2 Kopieren ausgewählter Komponenten (configtool/ecosconfig)
- 3 Erstellen einer Betriebssystembibliothek
- 4 Entwicklung der eigentlichen Anwendung
- 5 Kompilieren des Gesamtsystems



eCos-Entwicklungszyklus

- 1 Erstellen einer Konfiguration (configtool/ecosconfig)
- 2 Kopieren ausgewählter Komponenten (configtool/ecosconfig)
- 3 Erstellen einer Betriebssystembibliothek
- 4 Entwicklung der eigentlichen Anwendung
- 5 Kompilieren des Gesamtsystems



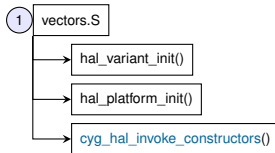
Wichtig!

Für jede Übung wird eine Konfiguration vorgegeben (Schritte 1–3)



1 vectors.S

- Hardwareinitialisierung
- Globale Konstruktoren

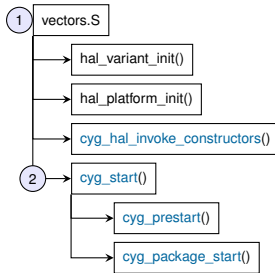


1 vectors.S

- Hardwareinitialisierung
- Globale Konstruktoren

2 cyg_start():

- Hardwareunabhängige Vorbereitungen



1 vectors.S

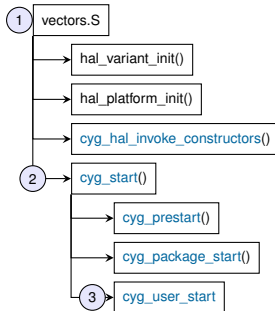
- Hardwareinitialisierung
- Globale Konstruktoren

2 cyg_start():

- Hardwareunabhängige Vorbereitungen

3 cyg_user_start:

- Einsprungpunkt für Anwendungscode!
- Erzeugen von Threads



1 vectors.S

- Hardwareinitialisierung
- Globale Konstruktoren

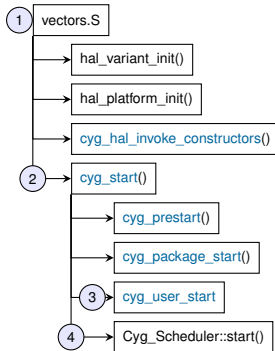
2 cyg_start():

- Hardwareunabhängige Vorbereitungen

3 cyg_user_start:

- Einsprungpunkt für Anwendungscode!
- Erzeugen von Threads

4 Starten des Schedulers



1 vectors.S

- Hardwareinitialisierung
- Globale Konstruktoren

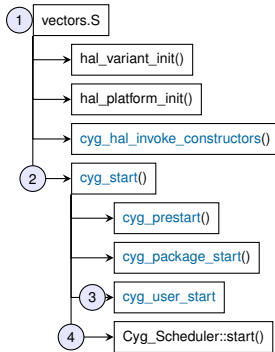
2 cyg_start():

- Hardwareunabhängige Vorbereitungen

3 cyg_user_start:

- Einsprungpunkt für Anwendungscode!
- Erzeugen von Threads

4 Starten des Schedulers

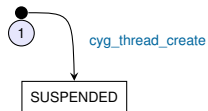


Wichtig!

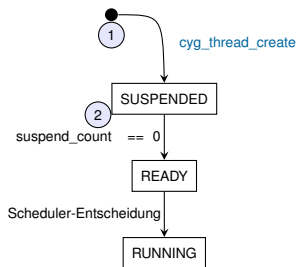
In allen Übungsaufgaben muss man `cyg_user_start()` implementieren und dort alle Threads anlegen. *Die Funktion muss zurückkehren!*



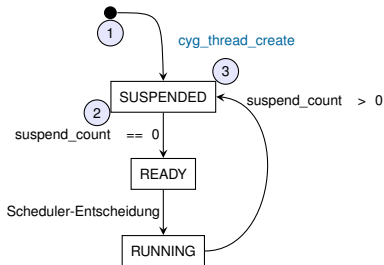
- 1 Thread wird im Zustand *suspended* erzeugt.
 - `suspend_count = 1`



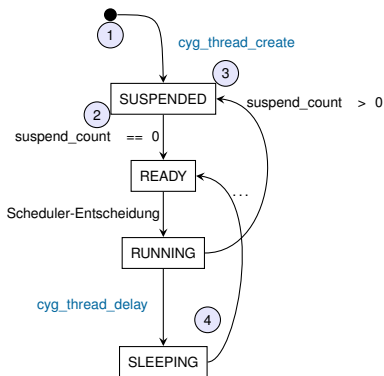
- 1 Thread wird im Zustand *suspended* erzeugt.
 - `suspend_count = 1`
- 2 `cyg_thread_resume` aktiviert
 - `suspend_count--`



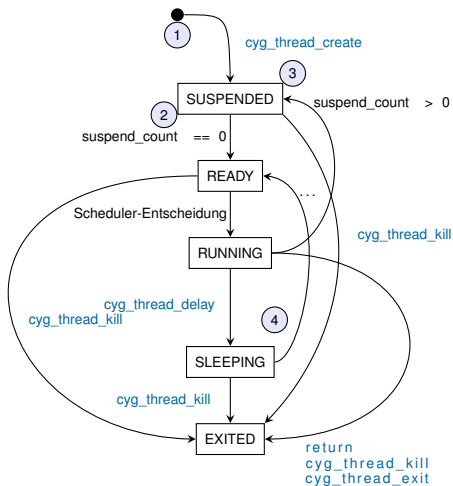
- 1 Thread wird im Zustand *suspended* erzeugt.
 - `suspend_count = 1`
- 2 `cyg_thread_resume` aktiviert
 - `suspend_count--`
- 3 `cyg_thread_suspend` suspendiert
 - `suspend_count++`



- 1 Thread wird im Zustand *suspended* erzeugt.
 - `suspend_count = 1`
- 2 `cyg_thread_resume` aktiviert
 - `suspend_count--`
- 3 `cyg_thread_suspend` suspendiert
 - `suspend_count++`
- 4 delay, mutex, semaphore wait



- 1 Thread wird im Zustand *suspended* erzeugt.
 - `suspend_count = 1`
- 2 `cyg_thread_resume` aktiviert
 - `suspend_count--`
- 3 `cyg_thread_suspend` suspendiert
 - `suspend_count++`
- 4 delay, mutex, semaphore wait
- 5 Threadterminierung



Threadderzeugung in eCos

```
1 #include <cyg/kernel/kapi.h>
2 void cyg_thread_create
3 (
4     cyg_addrword_t sched_info ,
5     cyg_thread_entry_t* entry ,
6     cyg_addrword_t entry_data ,
7     char* name ,
8     void* stack_base ,
9     cyg_ucount32 stack_size ,
10    cyg_handle_t* handle ,
11    cyg_thread* thread
12 );
```

- Einbinden der nötigen Headerdatei
- Anlegen eines neuen eCos-Threads



Threadderzeugung in eCos

```
1 #include <cyg/kernel/kapi.h>
2 void cyg_thread_create
3 (
4     cyg_addrword_t sched_info ,
5     cyg_thread_entry_t* entry ,
6     cyg_addrword_t entry_data ,
7     char* name ,
8     void* stack_base ,
9     cyg_ucount32 stack_size ,
10    cyg_handle_t* handle ,
11    cyg_thread* thread
12 );
```

- faktisch: Threadpriorität
- 0
 - höchste Priorität
 - für Systemprozesse freilassen
- CYG_THREAD_MIN_PRIORITY
 - Idle-Thread



Threadderzeugung in eCos

```
1 #include <cyg/kernel/kapi.h>
2 void cyg_thread_create
3 (
4     cyg_addrword_t sched_info ,
5     cyg_thread_entry_t* entry ,
6     cyg_addrword_t entry_data ,
7     char* name ,
8     void* stack_base ,
9     cyg_ucount32 stack_size ,
10    cyg_handle_t* handle ,
11    cyg_thread* thread
12 );
```

- Threadeinsprungpunkt
- Funktionszeiger
- Signatur:
`void (*)(cyg_addrword_t)`



```
1 #include <cyg/kernel/kapi.h>
2 void cyg_thread_create
3 (
4     cyg_addrword_t sched_info ,
5     cyg_thread_entry_t* entry ,
6     cyg_addrword_t entry_data ,
7     char* name ,
8     void* stack_base ,
9     cyg_ucount32 stack_size ,
10    cyg_handle_t* handle ,
11    cyg_thread* thread
12 );
```

- Threadparameter
- Beliebiger Übergabeparameter
 - z. B. Zeiger auf threadlokale Daten



Threadderzeugung in eCos

```
1 #include <cyg/kernel/kapi.h>
2 void cyg_thread_create
3 (
4     cyg_addrword_t sched_info ,
5     cyg_thread_entry_t* entry ,
6     cyg_addrword_t entry_data ,
7     char* name ,
8     void* stack_base ,
9     cyg_ucount32 stack_size ,
10    cyg_handle_t* handle ,
11    cyg_thread* thread
12 );
```

- Beliebiger Threadname
- Tipp: (gdb) info threads



```
1 #include <cyg/kernel/kapi.h>
2 void cyg_thread_create
3 (
4     cyg_addrword_t sched_info ,
5     cyg_thread_entry_t* entry ,
6     cyg_addrword_t entry_data ,
7     char* name ,
8     void* stack_base ,
9     cyg_ucount32 stack_size ,
10    cyg_handle_t* handle ,
11    cyg_thread* thread
12 );
```

- Basisadresse des Threadstacks
(→ &stack[0])
- `cyg_uint8`-Array
- Global definieren
→ Datensegment!
- *Warum ist die notwendig?*



Threadderzeugung in eCos

```
1 #include <cyg/kernel/kapi.h>
2 void cyg_thread_create
3 (
4     cyg_addrword_t sched_info ,
5     cyg_thread_entry_t* entry ,
6     cyg_addrword_t entry_data ,
7     char* name,
8     void* stack_base ,
9     cyg_ucount32 stack_size ,
10    cyg_handle_t* handle ,
11    cyg_thread* thread
12 );
```

■ Stackgröße in Bytes



```
1 #include <cyg/kernel/kapi.h>
2 void cyg_thread_create
3 (
4     cyg_addrword_t sched_info ,
5     cyg_thread_entry_t* entry ,
6     cyg_addrword_t entry_data ,
7     char* name ,
8     void* stack_base ,
9     cyg_ucount32 stack_size ,
10    cyg_handle_t* handle ,
11    cyg_thread* thread
12 );
```

- Eindeutiger Identifikator

- zur "Steuerung" z. B.:

`cyg_thread_resume(handle)`



```
1 #include <cyg/kernel/kapi.h>
2 void cyg_thread_create
3 (
4     cyg_addrword_t sched_info ,
5     cyg_thread_entry_t* entry ,
6     cyg_addrword_t entry_data ,
7     char* name ,
8     void* stack_base ,
9     cyg_ucount32 stack_size ,
10    cyg_handle_t* handle ,
11    cyg_thread* thread
12 );
```

- Speicher für interne Fadeninformationen
 - Fadenzustand u. a. suspend_count
- Vermeidung dynamischer Speicherallokation im Kernel



```
1 #include <cyg/kernel/kapi.h>
2 void cyg_thread_create
3 (
4     cyg_addrword_t sched_info ,
5     cyg_thread_entry_t* entry ,
6     cyg_addrword_t entry_data ,
7     char* name ,
8     void* stack_base ,
9     cyg_ucount32 stack_size ,
10    cyg_handle_t* handle ,
11    cyg_thread* thread
12 );
```

Ausführliche Dokumentation

<http://ecos.sourceware.org/docs-latest/ref/kernel-thread-create.html>



```
1 #include <cyg/hal/hal_arch.h>
2 #include <cyg/kernel/kapi.h>
3 #include <stdio.h>
4 #define MY_PRIORITY          11
5 #define STACKSIZE           (CYGNUM_HAL_STACK_SIZE_MINIMUM+4096)
6 static cyg_uint8            my_stack[STACKSIZE];
7 static cyg_handle_t         my_handle;
8 static cyg_thread           my_thread;
9
10 static void my_entry(cyg_addrword_t data) {
11     int message = (int) data;
12     ezs_printf("Beginning execution: thread data is %d\n", message);
13     for (;;) {
14         ezs_printf("Hello World!\n"); // \n flushes output
15         ezs_delay_us(1000000); // Delay for 1000000 * 1us = 1 second
16     }
17 }
18 void cyg_user_start(void) {
19     ezs_printf("Entering cyg_user_start() function\n");
20     cyg_thread_create(MY_PRIORITY, &my_entry, 0, "thread 1",
21                       my_stack, STACKSIZE, &my_handle, &my_thread);
22     cyg_thread_resume(my_handle); }
```



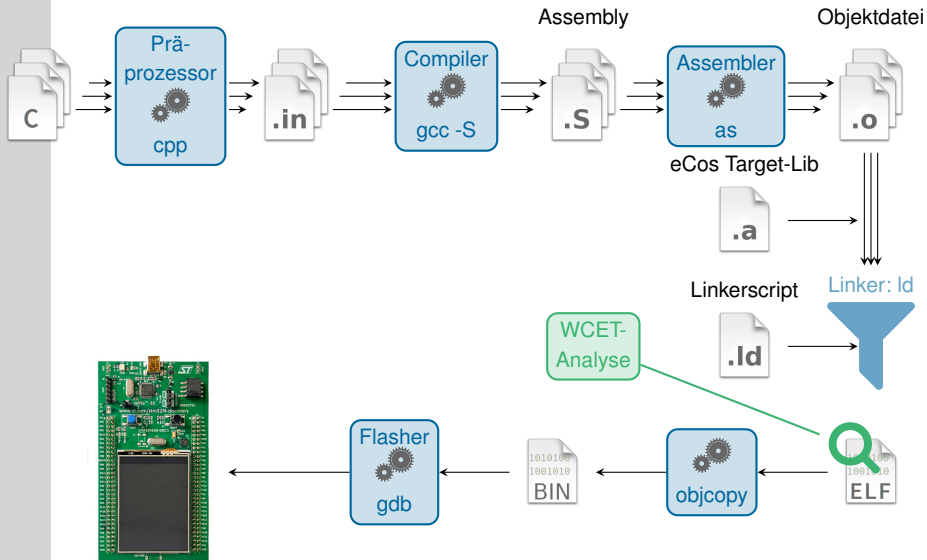
- 1 Einführung in eCos
- 2 Entwicklungsumgebung
 - Werkzeugkette und CMake
 - Blackmagic
 - Pulsweitenmodulation
 - Tiefpassfilter
 - Oszilloskop-Bedienung
- 3 Debuggen mit GDB



- 1 Einführung in eCos
- 2 Entwicklungsumgebung
 - **Werkzeugkette und CMake**
 - Blackmagic
 - Pulsweitenmodulation
 - Tiefpassfilter
 - Oszilloskop-Bedienung
- 3 Debuggen mit GDB



EZS-Toolchain



Wichtig!

Zu jeder Übungsaufgabe wird eine eCos-Konfiguration bereitgestellt.
Makefiles werden mit `cmake` generiert.

- 1 Vorgabe herunterladen, entpacken, Verzeichnis betreten
- 2 **Nötige Umgebungsvariablen setzen:** `source ecosenv.sh`
- 3 Eigene Quelldateien in `CMakeLists.txt` eintragen
- 4 `build` Verzeichnis betreten → out-of-source build²
- 5 Makefiles erzeugen: `cmake ..` (*cmake PUNKT PUNKT*)
- 6 Alles kompilieren: `make`
- 7 Flashen, ausführen, debuggen: `make flash`
~> Flasher & Debugger: `make gdb` (später ausführlicher)

²<https://gitlab.kitware.com/cmake/community/wikis/FAQ#out-of-source-build-trees>



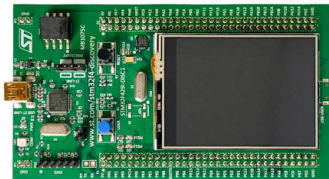
Wiki zum EZS-Board

<https://gitlab.cs.fau.de/ezs/ezs-board/wikis/home>

- Dokumentation im Wiki
 - 🔗 Hardware (EZS-Board)
 - 🔗 Entwicklungsumgebung
- Erweiterung durch *alle Teilnehmer an EZS* möglich
 - GitLab-Account notwendig



- ARM Cortex-M4 Prozessor
 - Flash-Speicher: 2 MB
 - RAM: 256 KB
- Umfangreiche Peripherie
 - Touch-LCD
 - Serielle Kommunikation
 - Timer
 - GPIOs
 - ADCs
 - 3-Achsen Gyroskop
 - Beschleunigungssensor
 - Audio Sensor
 - *integrierte Debugging-Schnittstelle*
 - ...



- 1 Einführung in eCos
- 2 **Entwicklungsumgebung**
 - Werkzeugkette und CMake
 - **Blackmagic**
 - Pulsweitenmodulation
 - Tiefpassfilter
 - Oszilloskop-Bedienung
- 3 Debuggen mit GDB





- Ausleihbare Boards sind mit Blackmagic Firmware³ ausgestattet
- Mini-USB-Kabel anschließen
- Nach Verbinden des USB-Kabels zwei serielle Ports, z. B.:
 - `/dev/ttyACM0`: Debugger
 - `/dev/ttyACM1`: serielle Kommunikation (Ausgabe z.B. mit cutecom)

³<https://github.com/blacksphere/blackmagic>



- Ausleihbare Boards sind mit Blackmagic Firmware³ ausgestattet
- Mini-USB-Kabel anschließen
- Nach Verbinden des USB-Kabels zwei serielle Ports, z. B.:
 - `/dev/ttyACM0`: Debugger
 - `/dev/ttyACM1`: serielle Kommunikation (Ausgabe z.B. mit cutecom)
- Exakte Ports zufällig \leadsto `/dev/serial/by-id/*Black_Magic*`
- 🔗 Für EZS: `/tmp/$USER-ezs-serial`

³<https://github.com/blacksphere/blackmagic>

- Bashprompt: %, gdb-Prompt: >

```
% cd <ezs-aufgabe1>
% source ecosenv.sh
% cd build
% cmake ..
% make
% arm-none-eabi-gdb -nh app.elf          # Starten des Debuggers
> target extended-remote /dev/ttyACM0  # gdb ueber ttyACM0
> monitor swd                          # Verwendung Serial Wire Debugging (SWD)
> attach 1                              # Erstes Interface verwenden
> load                                  # Laden des Systems in Flash (Flashen)
> continue                              # Starten der Ausfuehrung
```

- `gdb -nh`: Verhindert Ausführung der Befehle aus `~/.gdbinit`
- `gdb`-Befehle in Datei `flash.gdb` zusammenfassen und starten mittels:
`arm-none-eabi-gdb -batch -x flash.gdb -nh app.elf`



- Bashprompt: %, gdb-Prompt: >

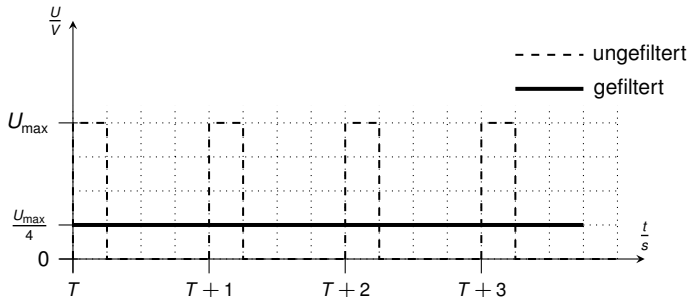
```
% cd <ezs-aufgabe1>
% source ecosenv.sh
% cd build
% cmake ..
% make
% arm-none-eabi-gdb -nh app.elf          # Starten des Debuggers
> target extended-remote /dev/ttyACM0 # gdb ueber ttyACM0
> monitor swd                          # Verwendung Serial Wire Debugging (SWD)
> attach 1                              # Erstes Interface verwenden
> load                                  # Laden des Systems in Flash (Flashen)
> continue                              # Starten der Ausfuehrung
```

- `gdb -nh`: Verhindert Ausführung der Befehle aus `~/gdbinit`
- gdb-Befehle in Datei `flash.gdb` zusammenfassen und starten mittels:
`arm-none-eabi-gdb -batch -x flash.gdb -nh app.elf`
- bereits implementiert in Target `make flash`



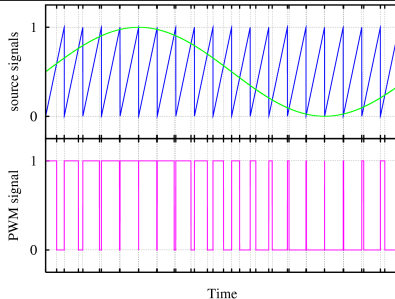
- 1 Einführung in eCos
- 2 **Entwicklungsumgebung**
 - Werkzeugkette und CMake
 - Blackmagic
 - **Pulsweitenmodulation**
 - Tiefpassfilter
 - Oszilloskop-Bedienung
- 3 Debuggen mit GDB





- 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)





Verfahren zur **Signalerzeugung**

- Hardware: Vergleich periodischer Zähler und Wert der Einschaltdauer
- Weit verbreitet: Motorsteuerung, Class-D-Verstärker, Schaltnetzteile, Nachrichtenübertragung,...
- Mittels Tiefpass \leadsto Digital-Analog-Wandlung
- libEZS: `void ezs_dac_write(uint8_t)` (zusätzlich *echter* DAC auf Board)
 \leadsto Wertebereich 0-255



- 1 Einführung in eCos
- 2 **Entwicklungsumgebung**
 - Werkzeugkette und CMake
 - Blackmagic
 - Pulsweitenmodulation
 - **Tiefpassfilter**
 - Oszilloskop-Bedienung
- 3 Debuggen mit GDB



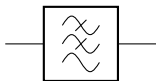


Abbildung: Schaltbild RC-Glied

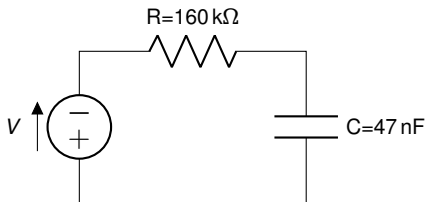


Abbildung: Schaltung RC-Filter auf Filterbaustein

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

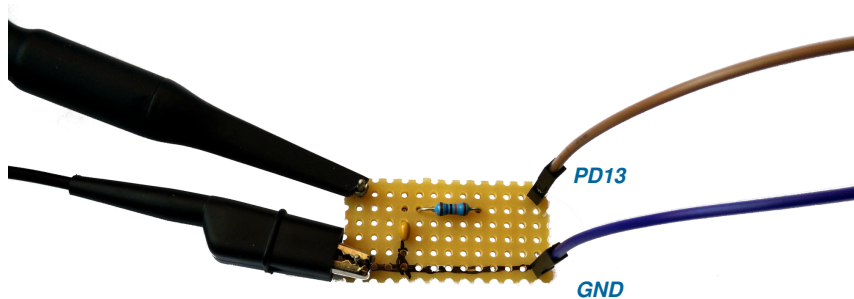
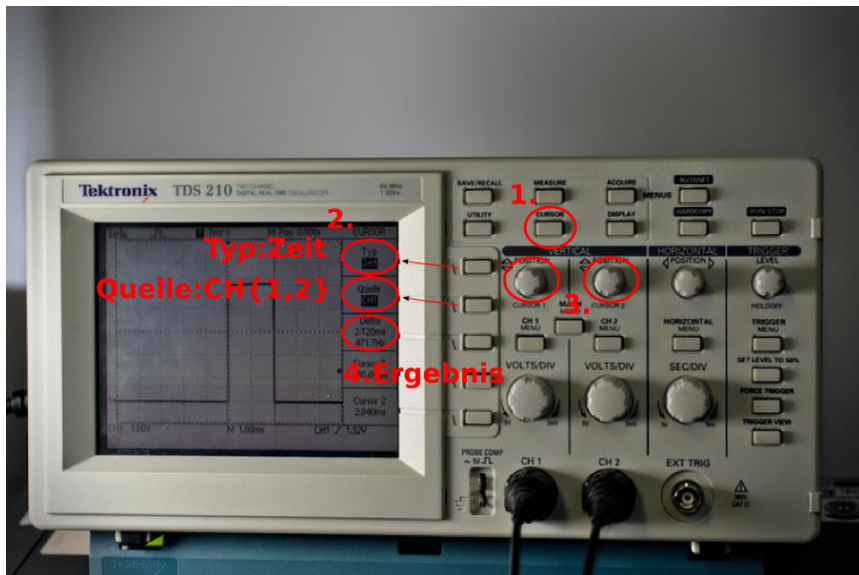


Abbildung: Filterbaustein anschließen, *schwarzer Strich markiert Masse*

- 1 Einführung in eCos
- 2 **Entwicklungsumgebung**
 - Werkzeugkette und CMake
 - Blackmagic
 - Pulsweitenmodulation
 - Tiefpassfilter
 - **Oszilloskop-Bedienung**
- 3 Debuggen mit GDB



Cursor



- 1 Einführung in eCos
- 2 Entwicklungsumgebung
 - Werkzeugkette und CMake
 - Blackmagic
 - Pulsweitenmodulation
 - Tiefpassfilter
 - Oszilloskop-Bedienung
- 3 Debuggen mit GDB



Aufrufen

- Interaktive gdb-Session:
% make gdb
- gdb-Dashboard:
% make debug
- Manueller Aufruf:
% arm-none-eabi-gdb \
-x ezs_dashboard.gdb app.elf
- Parameter -nh verwenden falls
.gdbinit vorhanden

Fenster

- 1 Source Code
- 2 Assembly
- 3 Stack
- 4 Threads
- 5 Lokale Variablen

```
File Edit View Search Terminal Help
and "show warranty" for details.
This GDB was configured as "--host=x86_64-pc-linux-gnu --target=arm-none-eabi".
Type "show configuration" for configuration details.
For bug reporting instructions, please see:
<http://www.gnu.org/software/gdb/bugs/>
Find the GDB manual and other documentation resources online at:
<http://www.gnu.org/software/gdb/documentation/>.
For help, type "help".
Type "apropos word" to search for commands related to "word"...
Reading symbols from ezs-aufgaben/ezs-aufgaben/HelloWorld/build/app.elf...done.
Target voltage: unknown
Available Targets:
No. Att Driver
1 STM32F4xx
--- Source
38 res_ps = (PRESCALER+1) * 1000000L / RCCLOCK;
39 res_us = res_ps / 1000000L;
40
41
42
43
44 cyg_uint64 ezs_counter_get(void) {
45     return timer_get_counter(TIM5);
46 }
47
48
49 cyg_uint64 ezs_counter_resolution_us(void){
50     return res_us;
51 }
52 }
--- Assembly
0x08000450 ezs_counter_get+0 push    {r3, lr}
0x08000452 ezs_counter_get+2 ldr     r0, [pc, #0] ; (0x080045c <ezs_counter_get()+12>)
0x08000454 ezs_counter_get+4 bl     0x0800f00 <timer_get_counter>
0x08000458 ezs_counter_get+8 movs   r1, #0
0x0800045a ezs_counter_get+10 pop    {r3, pc}
--- Stack
[0] from 0x08000452 in ezs_counter_get+2 at /home/noctux/work/ezs-aufgaben/ezs-aufgaben/HelloWorld/libEZ5/drivers/stm32f4/ezs_counter.cpp:41
(no arguments)
[1] from 0x0800044e in ezs_delay_us+110 at /home/noctux/work/ezs-aufgaben/ezs-aufgaben/HelloWorld/libEZ5/src/ezs_delay.c:15
arg microseconds = 1000
[2]
--- Threads
[1] id 0 from 0x08000452 in ezs_counter_get+2 at /home/noctux/work/ezs-aufgaben/ezs-aufgaben/HelloWorld/libEZ5/drivers/stm32f4/ezs_counter.cpp:41
--- Locals
ezs_counter_get () at /home/noctux/work/ezs-aufgaben/ezs-aufgaben/HelloWorld/libEZ5/drivers/stm32f4/ezs_counter.cpp:41
41     return timer_get_counter(TIM5);
Loading section .rom_vectors, size 0x8 lma 0x8000000
Loading section .ARM.exidx, size 0x8 lma 0x8000008
Loading section .text, size 0xd924 lma 0x8000010
Loading section .rodata, size 0x12a9 lma 0x8000938
Loading section .data, size 0xa0 lma 0x8000bbe8
Start address 0x8000010, load size 61564
Transfer rate: 16 KB/sec, 918 bytes/write.
>>> □
```



gdb Kommandos – I

Befehle haben Langformen (break) und Kurzformen (b)

Wichtige Befehle

- Breakpoint setzen:

»> b(reak) cyg_user_start

- Einzelschritt (Funktionen betreten):

»> s(tep)

- Einzelschritt

(Funktionen nicht betreten):

»> n(ext)

- Programm fortsetzen:

»> c(ontinue)

- Bis zum Ende der Funktion ausführen:

»> fin(ish)

- Funktion anzeigen:

»> l(ist) <funktionenname>

- gdb schließen:

»> q(uit) (oder Strg+D)

- Neu Flashen:

»> l(oad)

```
File Edit View Search Terminal Help
Type "show configuration" for configuration details.
For bug reporting instructions, please see:
<http://www.gnu.org/software/gdb/bugs/>.
Find the GDB manual and other documentation resources online at:
<http://www.gnu.org/software/gdb/documentation/>.
For help, type "help".
Type "apropos word" to search for commands related to "word"...
Reading symbols from ezs-aufgaben/ezs-aufgaben/HelloWorld/build/app.elf...done.
Target voltage: unknown
Available Targets:
No. Attc Driver
  3  STM32F4xx
-- Source
36 res_ps = (PRESCALER+1) * 100000L / RCCLOCK;
37 res_us = res_ps / 100000L;
38 }
39
40 cyg_uint64 ezs_counter_get(void) {
41     return timer_get_counter(TIM5);
42 }
43
44 cyg_uint64 ezs_counter_resolution_us(void){
45     return res_us;
46 }
-- Assembly
0x08000450 ezs_counter_get+0 push    {r3, lr}
0x08000452 ezs_counter_get+2 ldr     r0, [pc, #8] ; (0x080045c <ezs_counter_get()+12>)
0x08000454 ezs_counter_get+4 bl      #1 ; 0x0800f00 <timer_get_counter>
0x08000458 ezs_counter_get+8 movs   r1, #0
0x0800045a ezs_counter_get+10 pop    {r3, pc}
-- Stack
[6] from 0x08000452 in ezs_counter_get+2 at /home/noctux/work/ezs-aufgaben/ezs-aufgaben/HelloWorld/libEZS/drivers/stm32f4/ezs_counter.cpp:41
(no arguments)
[1] from 0x0800044e in ezs_delay_us+10 at /home/noctux/work/ezs-aufgaben/ezs-aufgaben/HelloWorld/libEZS/src/ezs_delay.c:15
arg microseconds = 1000
[1]
-- Threads
[1] id 0 from 0x08000452 in ezs_counter_get+2 at /home/noctux/work/ezs-aufgaben/ezs-aufgaben/HelloWorld/libEZS/drivers/stm32f4/ezs_counter.cpp:41
-- Locals
ezs_counter_get () at /home/noctux/work/ezs-aufgaben/ezs-aufgaben/HelloWorld/libEZS/drivers/stm32f4/ezs_counter.cpp:41
41     return timer_get_counter(TIM5);
Loading section .rom_vectors, size 0x8 lma 0x8000000
Loading section .ARM.exidx, size 0x8 lma 0x8000008
Loading section .text, size 0x924 lma 0x8000010
Loading section .rodata, size 0x12a8 lma 0x800d938
Loading section .data, size 0x4e8 lma 0x800e8e8
Start address 0x8000010, load size 61564
Transfer rate: 16 KB/sec, 918 bytes/write.
>>> break hello.c:40
Breakpoint 1 at 0x80000ec: file /home/noctux/work/ezs-aufgaben/ezs-aufgaben/HelloWorld/hello.c, line 40.
>>> □
```



Wichtige Befehle (Fortsetzung)

- Backtrace (Aufruf-Stack) anzeigen:
»» b(ack)t(race)
- Dashboard neu zeichnen:
»» dashboard
- Breakpoints anzeigen:
»» info breakpoints
- Breakpoint löschen:
»» delete <nummer>
- Variable anzeigen:
»» p(rint) <variablenname>

```
File Edit View Search Terminal Help
Source
35     int time_us = 0;
36     float daC_val = 0;
37
38     while(1)
39     {
40         if ((time_us/delay_us) % (100000/delay_us) == 0)
41             printf("Hallo Welt!\n");
42
43         up = not up;
44         ezs_gpio_set(up);
45
-- Assembly
0x080000e4 test_thread+20 ldr.w r8, [pc, #132] ; 0x080016c <test_thread+156>
0x080000e8 test_thread+24 ldr r7, [pc, #108] ; (0x0800158 <test_thread+136>)
0x080000ea test_thread+26 ldr r6, [pc, #112] ; (0x080015c <test_thread+140>)
0x080000ec test_thread+28 mov r0, r4
0x080000ee test_thread+30 asrs r1, r4, #31
0x080000f0 test_thread+32 mov.w r2, #1000 ; 0x3e8
0x080000f4 test_thread+36 movs r3, #0
-- Stack
[0] from 0x080000ec in test_thread+28 at /home/noctux/work/ezs-aufgaben/ezs-aufgaben/HelloWorld/hello.c:40
arg = <optimized out>
[1] from 0x08001072 in Cyg_HardwareThread::thread_entry+18 at /home/noctux/work/ezs-aufgaben/ezs-aufgaben/scripts/gen_14ezs/files/ecos/packages/kernel/cURRENT/src/common/thread_cxx:94
arg thread = 0x20000730 <threaddata>
[1]
-- Threads
[1] id 0 from 0x080000ec in test_thread+28 at /home/noctux/work/ezs-aufgaben/ezs-aufgaben/HelloWorld/hello.c:40
-- Locals
arg = <optimized out>
up = false
time_us = 0
daC_val = <optimized out>
>>> info breakpoints
Num Type Breakpoint Disp Enb Address What
1 breakpoint keep y 0x080000ec in test_thread at /home/noctux/work/ezs-aufgaben/ezs-aufgaben/HelloWorld/hello.c:40
2 breakpoint already hit 1 time
2 breakpoint keep y 0x08000170 in cyg_user_start at /home/noctux/work/ezs-aufgaben/ezs-aufgaben/HelloWorld/hello.c:55
>>> [ ]
```



- gdb-Dashboard benötigt einen gdb mit python-Bindings
- gdb „abschießen“:⁴
`% killall gdb-multiarch -s SIGKILL`
- EZS-Board in initialen Zustand setzen:
USB-Kabel abstecken & wieder anstecken
- GDB-Spickzettel:
<http://darkdust.net/files/GDB%20Cheat%20Sheet.pdf>

GDB-Einführung aus BS

https://www4.cs.fau.de/Lehre/WS19/V_BS/Uebungen/seminar-gdb.pdf

⁴gdb-multiarch ist der Name des gdb-Binaries im CIP, kann bei euch zu Hause abweichen

Übersicht hilfreicher make-Targets

- `flash` Baut Anwendung und schreibt sie auf das Board.
- `gdb` Startet eine interaktiver GDB-Sitzung.
- `debug` Startet eine GDB-Sitzung mit dem EZS-Dashboard.
- `doc` Erstellt Dokumentation für die von uns bereitgestellten Funktionen.
- `sanity-test` Testet grundlegende Funktionalität der Aufgabe.
- `submit` Erzeugt eine Abgabe (*rechtzeitig* aufrufen).
- `diff` Zeigt Unterschiede zwischen dem aktuellen Stand und der letzten Abgabe.



Besprechung der Übungsaufgabe

„Hallo Welt!“



42

