

# Echtzeitsysteme

Übungen zur Vorlesung

Entwicklungsumgebung

**Simon Schuster   Phillip Raffeck**

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

Wintersemester 2019/20



Schu, PR   EZS (WS19)

1/35

## Übersicht

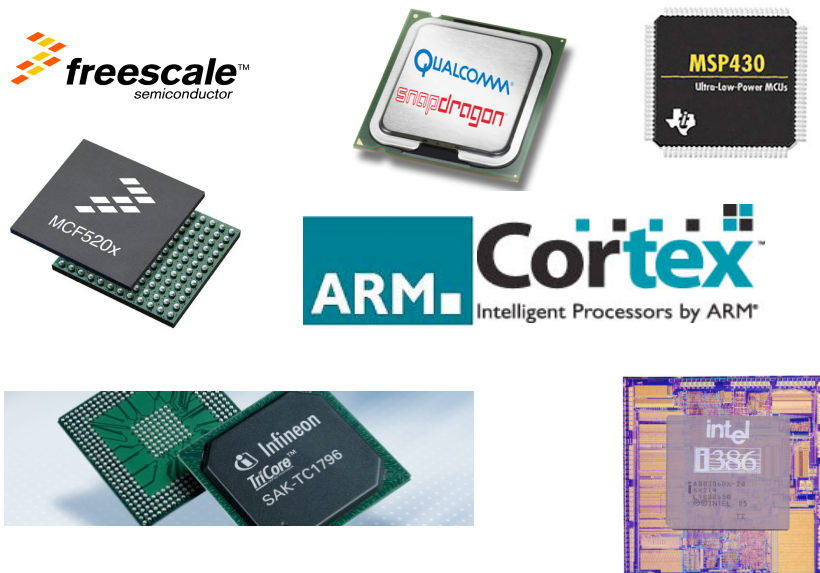
- 1 Einführung in eCos
- 2 Entwicklungsumgebung
  - Werkzeugkette und CMake
  - Blackmagic
  - Pulsweitenmodulation
  - Tiefpassfilter
  - Oszilloskop-Bedienung
- 3 Debuggen mit GDB



Schu, PR   EZS (WS19)

2/35

## Prozessorvielfalt in der Echtzeitwelt



Schu, PR   EZS (WS19)  
1 Einführung in eCos

3/35

## Noch mehr Betriebssysteme



Schu, PR   EZS (WS19)  
1 Einführung in eCos

4/35

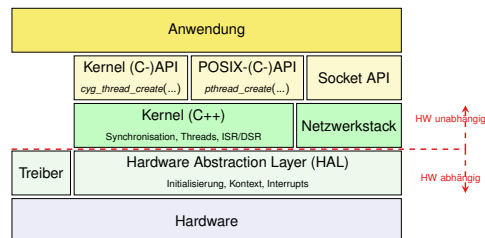
*eCos is an embedded, highly configurable, open-source, royalty-free, real-time operating system.*

- 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)

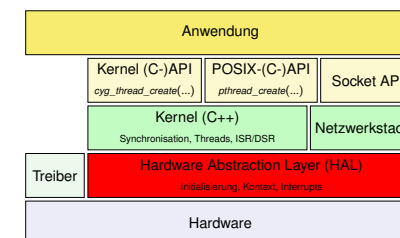
<http://ecos.sourceforge.org>



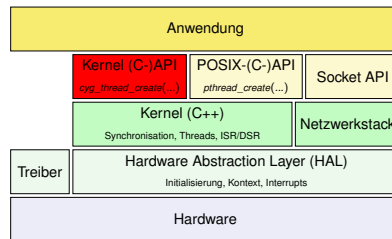
- Fujitsu SPARClite
- Matsushita MN10300
- Motorola PowerPC
- Advanced RISC Machines (ARM)
- Toshiba TX39
- Infineon TriCore
- Hitachi SH3
- NEC VR4300
- MB8683X
- ARM Cortex
- Intel x86
- ...



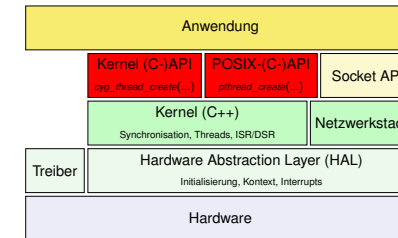
- Abstrahiert CPU- und plattformspezifische Eigenschaften
  - Kontextwechsel
  - Interruptverwaltung
  - CPU-Erkennung, Startup
  - Zeitgeber, I/O-Registerzugriffe



- Implementiert in C++
- Feingranular konfigurierbar
  - Verschiedene Schedulingstrategien (Bitmap/Multilevel Queue)
  - Zeitscheibenbasiert, präemptiv, prioritätenbasiert
- Verschiedene Synchronisationsstrategien
  - Mutexe, Semaphore, Bedingungsvariablen
  - Messages Boxes

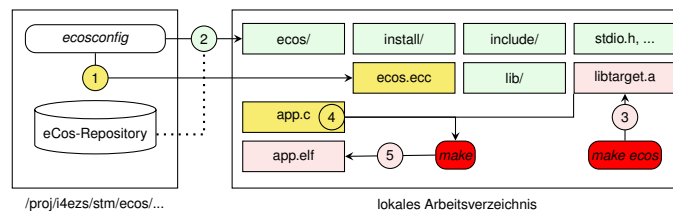


- Kernel API
  - vollständige C-Schnittstelle
  - siehe Dokumentation<sup>1</sup>
- (Optionale) POSIX-Kompatibilitätsschicht
  - Scheduling-Konfiguration, `pthread_*`
  - Timer, Semaphore, Message Queues, Signale, ...



## 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



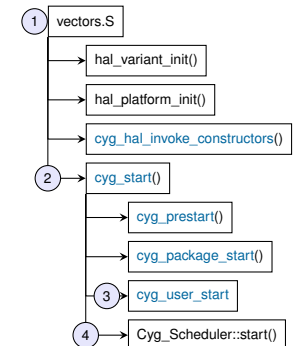
**Wichtig!**

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



## eCos-Systemstart

- 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!**

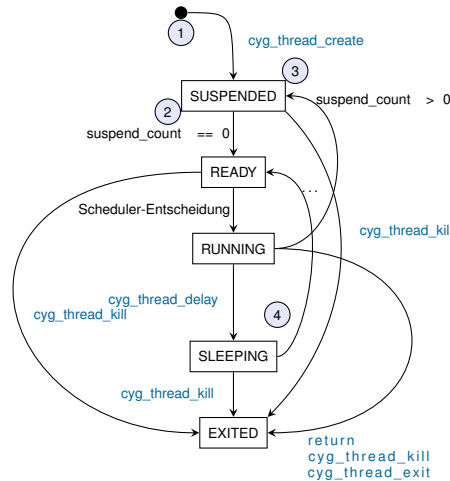
In allen Übungsaufgaben muss man `cyg_user_start()` implementieren und dort alle Threads anlegen. *Die Funktion muss zurückkehren!*



## eCos-Threads

### Threadzustände und Übergänge

- 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



## Threaderzeugung 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

### Ausführliche Dokumentation

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



## Threaderzeugung 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
- CYG\_THREAD\_MIN\_PRIORITY
  - Idle-Thread

### Ausführliche Dokumentation

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



## Threaderzeugung 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)`

### Ausführliche Dokumentation

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



## Threaderzeugung 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 );
```

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

### Ausführliche Dokumentation

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



## Threaderzeugung 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

### Ausführliche Dokumentation

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



## Threaderzeugung 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 );
```

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

### Ausführliche Dokumentation

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



## Threaderzeugung 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

### Ausführliche Dokumentation

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



## Threaderzeugung 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 );
```

- Eindeutiger Identifikator
  - zur "Steuerung" z. B.:  
`cyg_thread_resume(handle)`

### Ausführliche Dokumentation

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



## Threaderzeugung 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 );
```

- Speicher für interne Fadeninformationen
  - Fadenzustand u. a.  
suspend\_count
- Vermeidung dynamischer Speicherallokation im Kernel

### Ausführliche Dokumentation

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



## Threaderzeugung 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 );
```

### Ausführliche Dokumentation

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



## eCos-Beispielanwendung

```
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); }
```



## Übersicht

### 1 Einführung in eCos

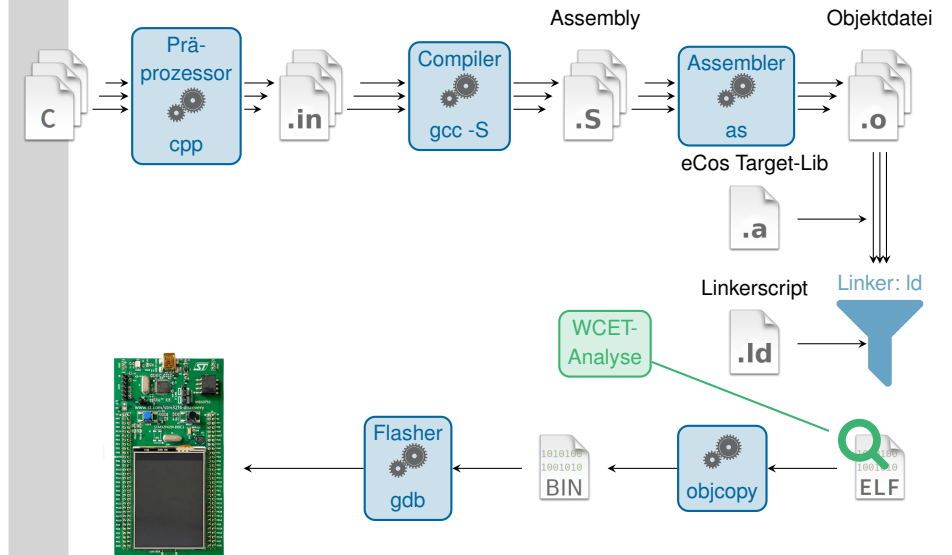
### 2 Entwicklungsumgebung

- Werkzeugkette und CMake
- Blackmagic
- Pulsweitenmodulation
- Tiefpassfilter
- Oszilloskop-Bedienung

### 3 Debuggen mit GDB



## EZS-Toolchain



## EZS-Entwicklungsumgebung

### 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 build<sup>2</sup>
- 5 Makefiles erzeugen: `cmake ..` (*cmake PUNKT PUNKT*)
- 6 Alles kompilieren: `make`
- 7 Flashen, ausführen, debuggen: `make flash`  
→ Flasher & Debugger: `make gdb` (später ausführlicher)



## Dokumentation zum EZS-Board

### Wiki zum EZS-Board

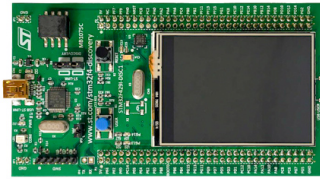
<https://gitlab.cs.fau.de/ezs/ezs-board/wikis/home>

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

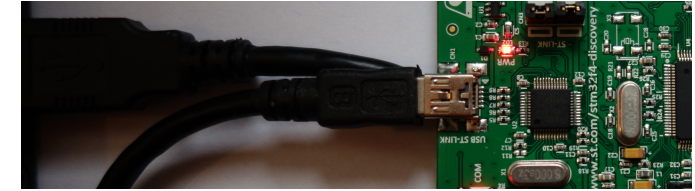


## Entwicklungsplattform STM32F429

- 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*
  - ...



## Flashen & Debuggen: Blackmagic Firmware



- Ausleihbare Boards sind mit Blackmagic Firmware<sup>3</sup> 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`



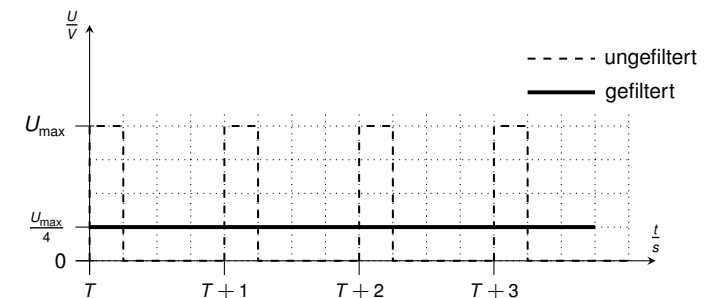
## Flashen mittels Kommandozeile

- 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`



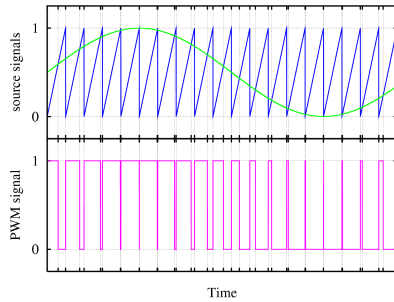
## Digital-Analog Umwandlung & Pulsweitenmodulation I



- 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  $\leadsto$  Digital-Analog-Wandlung
- libEVS: `void ezs_dac_write(uint8_t)` (zusätzlich *echter* DAC auf Board)



## Tiefpassfilter

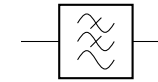


Abbildung: Schaltbild RC-Glied

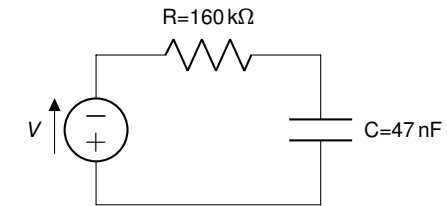


Abbildung: Schaltung RC-Filter auf Filterbaustein

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



## Tiefpassfilter anschließen

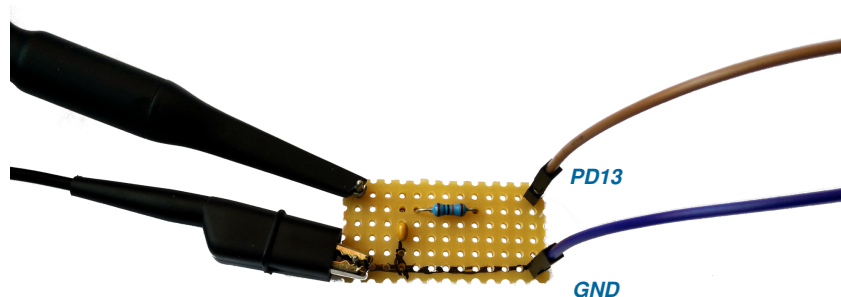
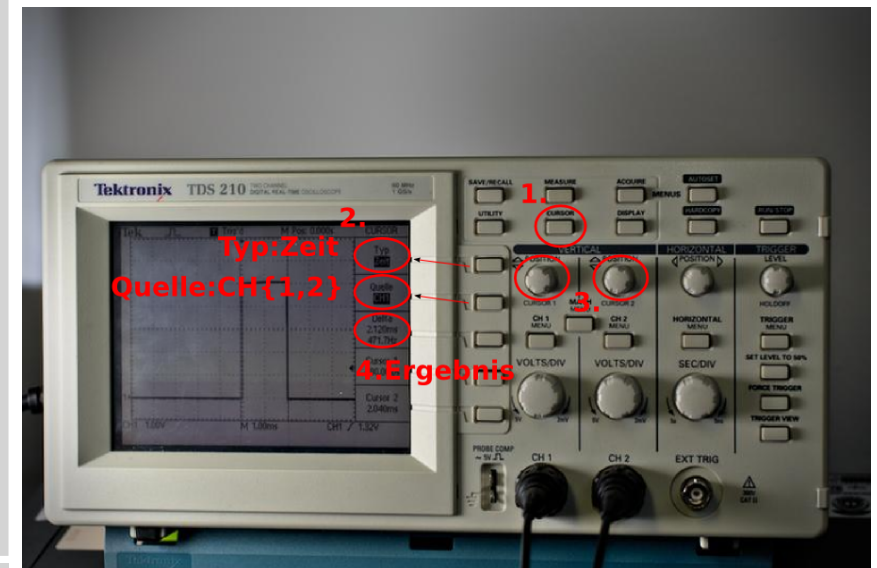


Abbildung: Filterbaustein anschließen, *schwarzer Strich markiert Masse*



## Cursor



# Übersicht

## 1 Einführung in eCos

## 2 Entwicklungsumgebung

- Werkzeugkette und CMake
- Blackmagic
- Pulsweitenmodulation
- Tiefpassfilter
- Oszilloskop-Bedienung

## 3 Debuggen mit GDB



Schu, PR  
EZS (WS19)  
3 Debuggen mit GDB

28/35

# gdb-Dashboard

## Aufrufen

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

## Fenster

- Source Code
- Assembly
- Stack
- Threads
- Lokale Variablen

```
File Edit View Search Terminal Help
nd "show warranty" for details.
This GDB was configured as "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/helloWorld/build/app.elf...done.
Target voltage: unknown
Available targets:
No. Att Driver
1 ST0274xx
-- Source
26 res_ps = (PRESCALER+1) * 1000000L / RCCCLK;
27 res_us = res_ps / 1000000L;
28
29
30 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
0x0000450 ezs_counter_get+0 push    {r2, lr}
0x0000452 ezs_counter_get+2 ldr     r0, [pc, #8] ; (0x000045c <ezs_counter_get()+12>)
0x0000454 ezs_counter_get+4 bl      0x0000700 <timer_get_counter>
0x0000456 ezs_counter_get+6 movs    r1, #0
0x0000458 ezs_counter_get+8 pop     {r2, pc}
-- Stack
[8] from 0x0000452 in ezs_counter_get+2 at /home/noctux/work/ezs-aufgaben/ezs-aufgaben/helloWorld/LibEZS/drivers/sta3274/ezs_counter.c:41
(no arguments)
[1] from 0x000044e in ezs_delay_us+10 at /home/noctux/work/ezs-aufgaben/ezs-aufgaben/helloWorld/LibEZS/src/ezs_delay.c:13
arg microseconds = 1000
[1]
-- Threads
[1] id 0 from 0x0000452 in ezs_counter_get+2 at /home/noctux/work/ezs-aufgaben/ezs-aufgaben/helloWorld/LibEZS/drivers/sta3274/ezs_counter.c:41
Locals
ezs_counter_get() at /home/noctux/work/ezs-aufgaben/ezs-aufgaben/helloWorld/LibEZS/drivers/sta3274/ezs_counter.c:41
41     return timer_get_counter(TIM5);
Loading section .rom_vectors, size 0x8 lma 0x0000000
Loading section .ARM.exidx, size 0x8 lma 0x0000008
Loading section .text, size 0xd24 lma 0x0000010
Loading section .rodata, size 0x12a0 lma 0x0000038
Loading section .data, size 0x40 lma 0x00000b8
Start address 0x000010, load size 61504
Transfer rate: 16 KB/sec, 918 bytes/write.
>>> [
```



Schu, PR  
EZS (WS19)  
3 Debuggen mit GDB

29/35

# 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) < Funktionsname>
- gdb schließen:  
>>> q(uit) (oder Strg+D)
- Neu Flashen:  
>>> l(od)

```
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/helloWorld/build/app.elf...done.
Target voltage: unknown
Available targets:
No. Att Driver
1 ST0274xx
-- Source
26 res_ps = (PRESCALER+1) * 1000000L / RCCCLK;
27 res_us = res_ps / 1000000L;
28
29
30 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
0x0000450 ezs_counter_get+0 push    {r2, lr}
0x0000452 ezs_counter_get+2 ldr     r0, [pc, #8] ; (0x000045c <ezs_counter_get()+12>)
0x0000454 ezs_counter_get+4 bl      0x0000700 <timer_get_counter>
0x0000456 ezs_counter_get+6 movs    r1, #0
0x0000458 ezs_counter_get+8 pop     {r2, pc}
-- Stack
[8] from 0x0000452 in ezs_counter_get+2 at /home/noctux/work/ezs-aufgaben/ezs-aufgaben/helloWorld/LibEZS/drivers/sta3274/ezs_counter.c:41
(no arguments)
[1] from 0x000044e in ezs_delay_us+10 at /home/noctux/work/ezs-aufgaben/ezs-aufgaben/helloWorld/LibEZS/src/ezs_delay.c:13
arg microseconds = 1000
[1]
-- Threads
[1] id 0 from 0x0000452 in ezs_counter_get+2 at /home/noctux/work/ezs-aufgaben/ezs-aufgaben/helloWorld/LibEZS/drivers/sta3274/ezs_counter.c:41
Locals
ezs_counter_get() at /home/noctux/work/ezs-aufgaben/ezs-aufgaben/helloWorld/LibEZS/drivers/sta3274/ezs_counter.c:41
41     return timer_get_counter(TIM5);
Loading section .rom_vectors, size 0x8 lma 0x0000000
Loading section .ARM.exidx, size 0x8 lma 0x0000008
Loading section .text, size 0xd24 lma 0x0000010
Loading section .rodata, size 0x12a0 lma 0x0000038
Loading section .data, size 0x40 lma 0x00000b8
Start address 0x000010, load size 61504
Transfer rate: 16 KB/sec, 918 bytes/write.
>>> break hello.c:40
Breakpoint 1 at 0x000000ec: file /home/noctux/work/ezs-aufgaben/ezs-aufgaben/helloWorld/hello.c, line 40.
>>> [
```



Schu, PR  
EZS (WS19)  
3 Debuggen mit GDB

30/35

# gdb Kommandos – II

## 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) % (1000000/delay_us) == 0)
41     {
42         printf("Hello Welt\n");
43         up = not up;
44         ezs_gpio_set(up);
45     }
46 }
-- Assembly
0x000000e0 test_thread+20 ldr     r8, [pc, #132] ; 0x000010c <test_thread+156>
0x000000e2 test_thread+22 ldr     r6, [pc, #108] ; (0x000010c <test_thread+156>)
0x000000e4 test_thread+24 ldr     r6, [pc, #112] ; (0x000010c <test_thread+156>)
0x000000e6 test_thread+26 mov     r4, r4
0x000000e8 test_thread+28 asrs    r2, r4, #1
0x000000ea test_thread+30 mov     r2, #1000
0x000000ec test_thread+32 mov     r2, #1000
0x000000ee test_thread+34 movs    r2, #0
-- Stack
[8] from 0x000000ec in test_thread+32 at /home/noctux/work/ezs-aufgaben/ezs-aufgaben/helloWorld/hello.c:40
arg = <optimized out>
[1] from 0x00000010 in Sys_HardwareThread:thread_entry+18 at /home/noctux/work/ezs-aufgaben/ezs-aufgaben/helloWorld/hello.c:40
arg thread = 0x00000010
arg thread = 0x00000010
arg thread = 0x00000010
-- Threads
[1] id 0 from 0x000000ec in test_thread+32 at /home/noctux/work/ezs-aufgaben/ezs-aufgaben/helloWorld/hello.c:40
arg = <optimized out>
arg = false
time_us = 0
dac_val = <optimized out>
-- info breakpoints
Num Type Disposition What
1 breakpoint keep y 0x000000ec in test_thread at /home/noctux/work/ezs-aufgaben/ezs-aufgaben/helloWorld/hello.c:40
2 breakpoint already hit 1 time
3 breakpoint keep y 0x00000170 in cyg_user_start at /home/noctux/work/ezs-aufgaben/ezs-aufgaben/helloWorld/hello.c:55
>>> [
```



Schu, PR  
EZS (WS19)  
3 Debuggen mit GDB

31/35

## Weiterführende Informationen

- gdb-Dashboard benötigt einen gdb mit python-Bindings
- gdb „abschießen“:<sup>4</sup>  
% 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](https://www4.cs.fau.de/Lehre/WS19/V_BS/Uebungen/seminar-gdb.pdf)



<sup>4</sup>gdb-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.

