

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>

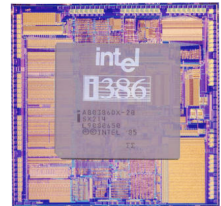
19.10.2018



- 1 Einführung in eCos
- 2 Entwicklungsumgebung
 - Pulsweitenmodulation
 - Tiefpassfilter
 - Oszilloskop-Bedienung
 - Auflösung von Zeiten in eCos
- 3 Debuggen mit GDB

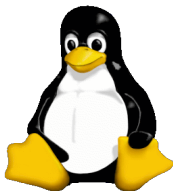


Prozessorvielfalt in der Echtzeitwelt



free **RTOS**

μClinux



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>

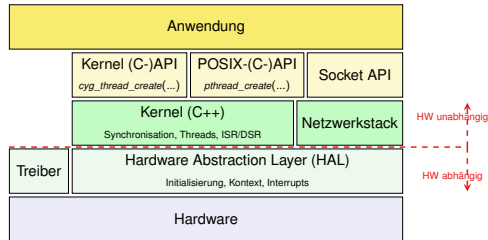


Unterstützte Plattformen

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

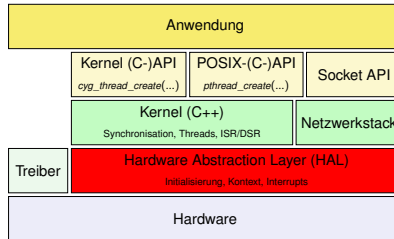
- Fujitsu SPARClite
- Matsushita MN10300
- Motorola PowerPC
- Advanced RISC Machines (ARM)
- Toshiba TX39
- Infineon TriCore
- Hitachi SH3
- NEC VR4300
- MB8683X
- ☞ *ARM Cortex*
- ☞ *Intel x86*
- ...



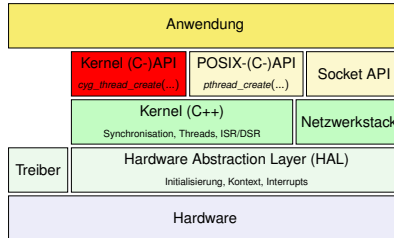


■ Abstrahiert CPU- und plattformspezifische Eigenschaften

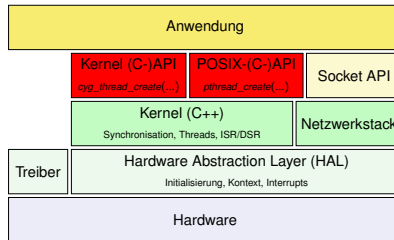
- Kontextwechsel
- Interruptverwaltung
- CPU-Erkennung, Startup
- Zeitgeber, I/O-Registerzugriffe



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

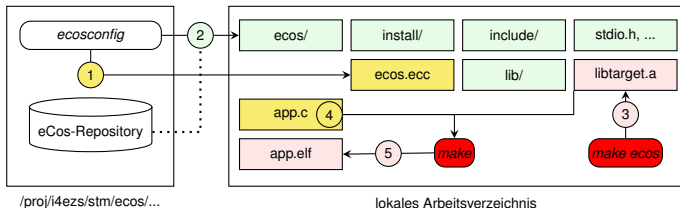


- Kernel API
 - vollständige C-Schnittstelle
 - siehe Dokumentation¹
- (Optionale) POSIX-Kompatibilitätsschicht
 - Scheduling-Konfiguration, *pthread*_*
 - Timer, Semaphore, Message Queues, Signale, ...



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

- 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

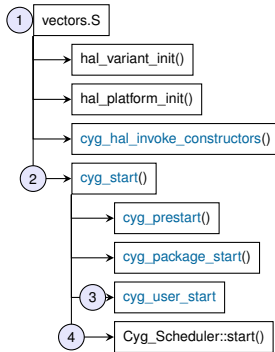
2 cyg_start():

- Hardwareunabhängige Vorbereitungen

3 cyg_user_start:

- Einsprungpunkt für Anwendungscode!
- Erzeugen von Threads

4 Starten des Schedulers

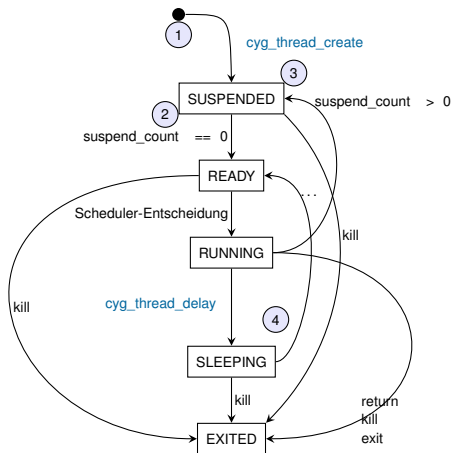


Wichtig!

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`
- 2 `cyg_thread_resume` aktiviert
 - `suspend_count--`
- 3 `cyg_thread_suspend` suspendiert
 - `suspend_count++`
- 4 bereit
- 5 delay, mutex, semaphore wait
- 6 Threadterminierung



```
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.sourceware.org/docs-latest/ref/kernel-thread-create.html>



```
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 );
```

- Scheduling-Informationen
- z. B. MLQ-Scheduler
 - Threadpriorität
 - Datentyp `cyg_uint8`

Ausführliche Dokumentation

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



```
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 );
```

- Thread Einsprungpunkt
- Funktionszeiger
- Signatur:
`void (*)(cyg_addrword_t)`

Ausführliche Dokumentation

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




```
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 );
```

- Thread Parameter
- Beliebige Übergabeparameter
 - z. B. Zeiger auf threadlokale Daten

Ausführliche Dokumentation

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



```
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>



```
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>



```
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>



```
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>



```
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>



```
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

- Pulsweitenmodulation
- Tiefpassfilter
- Oszilloskop-Bedienung
- Auflösung von Zeiten in eCos

3 Debuggen mit GDB



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)

²www.cmake.org/Wiki/CMake_FAQ#Out-of-source_build_trees

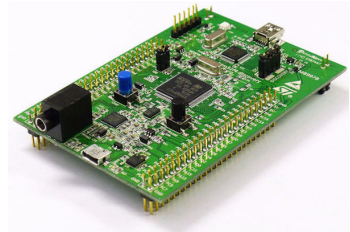
Wiki zum EZS-Board

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

- Umstellung auf neues Entwicklungsboard
 - ↪ Mögliche Probleme
- Nicht unterstützte Plattform
- 👉 Dokumentation im Wiki
- Erweiterung durch *alle Teilnehmer an EZS*
 - gitlab account notwendig
- Besondere Aufgaben
 - Anleitung „EZS-Board unter Windows“
 - Anleitung „EZS-Board unter macOS“
- 👉 Gutscheine für I4-Kaffeekarte



- ARM Cortex-M4 Prozessor
 - Flash-Speicher: 512 KB
 - RAM: 128 KB
- Umfangreiche Peripherie
 - Serielle Kommunikation
 - Timer
 - GPIOs
 - ADCs
 - 3-Achsen Gyroskop
 - Beschleunigungssensor
 - Audio Sensor
 - *integrierte Debugging-Schnittstelle*
 - ...





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

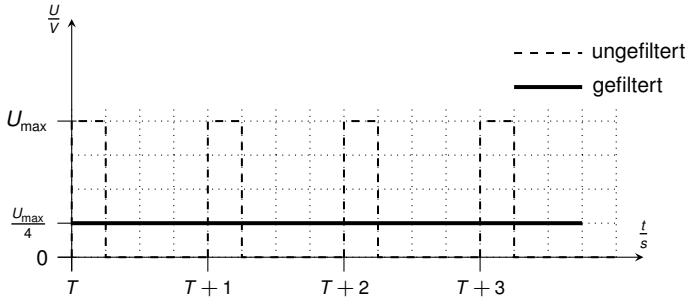
³<https://github.com/blacksphere/blackmagic>

- 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
```

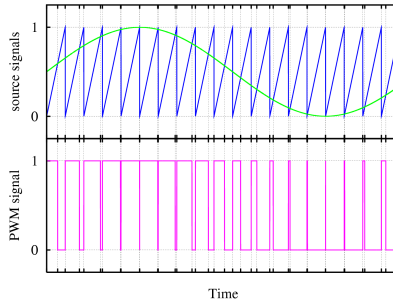
- gdb -nh: Verhindert Ausführung der Befehle aus ~/.gdbinit
- gdb-Befehle in Datei flash.gdb zusammenfassen und starten mittels:
arm-none-eabi-gdb --batch -x flash.gdb -nh app.elf





- 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
- libEZX: `void ezs_dac_write(uint8_t)` (zusätzlich *echter* DAC auf Board)



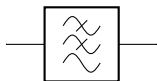


Abbildung: Schaltbild RC-Glied

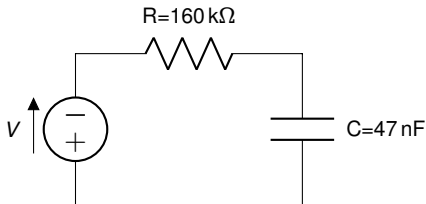
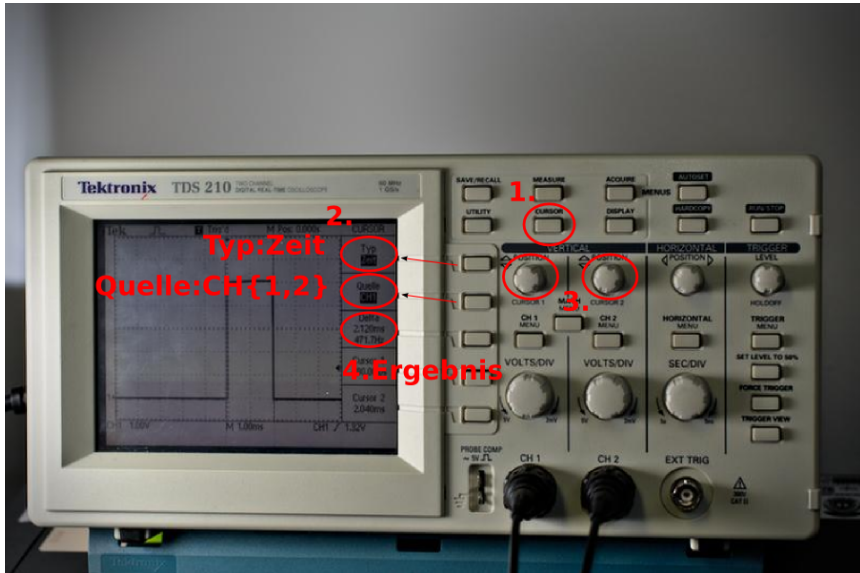


Abbildung: Schaltung RC-Filter auf EZS-Board

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





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

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



- 1 Einführung in eCos
- 2 Entwicklungsumgebung
 - Pulsweitenmodulation
 - Tiefpassfilter
 - Oszilloskop-Bedienung
 - Auflösung von Zeiten in eCos
- 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
36 res_ps = (PRESCALER+1) * 1000000L / RCCCLK;
37 res_us = res_ps / 1000000L;
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 {r7, lr}
0x08000452 ezs_counter_get+2 ldr lr, {0, #0} ; (0x080045c <ezs_counter_get()+12>)
0x08000454 ezs_counter_get+4 bl 0x0800700 <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/libEZS/drivers/stm32f4/ezs_counter.cpp:41
(no arguments)
[1] from 0x0800044b in ezs_delay_us+110 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 0x8000938
Loading section .data, size 0x4a0 lma 0x8000be8
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(step)
```

- Einzelschritt

(Funktionen nicht betreten):

```
>>> n(ext)
```

- Programm fortsetzen:

```
>>> c(ontinue)
```

- Bis zum Ende der Funktion

ausführen:

```
>>> fin(ish)
```

- Funktion anzeigen:

```
>>> l(list) <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/ezs-aufgaben/HelloWorld/build/app.elf...done.
Target voltage: unknown
Available Targets:
No. Att Driver
  1. STM32F4xx

-- Source
36 res_ps = (PRESCALER+1) * 1000000L / RCCCLK;
37 res_us = res_ps / 1000000L;
38 }
39
40 cyg_uint64 ezs_counter_get(void) {
41     return timer_get_counter(TIM5);
42 }
43
44 cy_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] ; (0x800045c <ezs_counter_get()+12>)
0x08000454 ezs_counter_get+4 bl     0x08000400 <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 0x080004e6 in ezs_delay_us+110 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
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 0x12e lma 0x800d93b
Loading section .data, size 0x4e lma 0x800debe6
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) % (1000000/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] ; 0x0800016c <test_thread+156>
0x080000e8 test_thread+24 ldr r7, [pc, #108] ; (0x08000158 <test_thread+136>)
0x080000ea test_thread+26 ldr r6, [pc, #112] ; (0x0800015c <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-aufgabe
n/scripts/gen_14ezs/Files/ecos/packages/kernel/curent/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/hel
lo.c:40
-- Locals
arg = <optimized out>
up = false
time_us = 0
daC_val = <optimized out>

>>> info breakpoints
Num Type 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
ben breakpoint keep y 0x08000170 in cyg_user_start at /home/noctux/work/ezs-aufgaben/ezs-aufga
ben/HelloWorld/hello.c:55
>>>
```



- gdb-Dashboard benötigt einen gdb mit python-Bindings
- gdb „abschießen“:
`% killall arm - none - eabi - gdb -s SIGKILL`
- EZS-Board in initialen Zustand setzen:
USB-Kabel abstecken & wieder anstecken
- GDB Spickzettel:
<http://darkdust.net/files/GDB%20Cheat%20Sheet.pdf>

