

Echtzeitsysteme

Übungen zur Vorlesung

Entwicklungsumgebung

Florian Schmaus Peter Wägemann

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

19.10.2017



FS, PW EZS (19.10.2017)

1/29

Übersicht

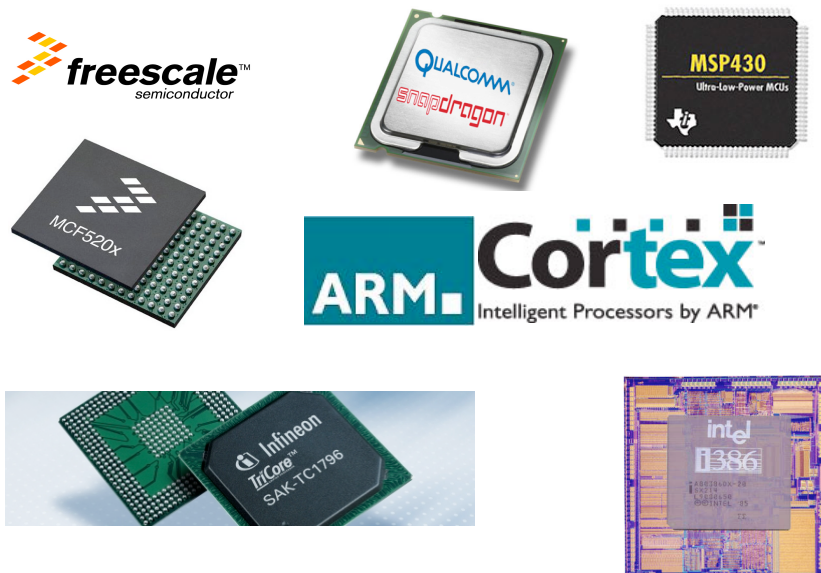
- 1 Einführung in eCos
- 2 Entwicklungsumgebung
- 3 Debuggen mit GDB



FS, PW EZS (19.10.2017)

2/29

Prozessorvielfalt in der Echtzeitwelt



FS, PW EZS (19.10.2017)
1 Einführung in eCos

3/29

Noch mehr Betriebssysteme



FS, PW EZS (19.10.2017)
1 Einführung in eCos

4/29

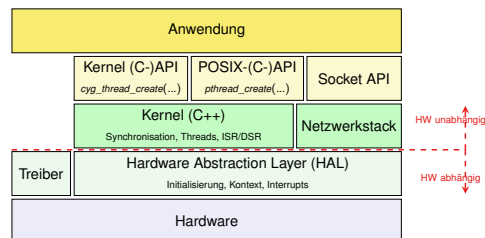
eCos is an embedded, highly configurable, open-source, royalty-free, real-time operating system.

- 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)

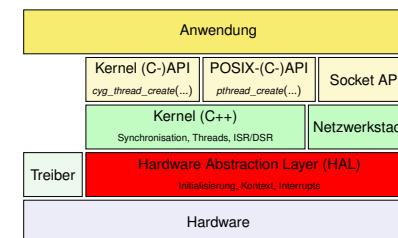
<http://ecos.sourceforge.org>



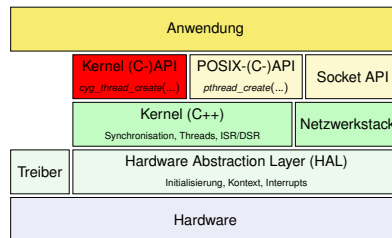
- Fujitsu SPARClite
- Matsushita MN10300
- Motorola PowerPC
- Advanced RISC Machines (ARM)
- Toshiba TX39
- Infineon TriCore
- Hitachi SH3
- NEC VR4300
- MB8683X
- ARM Cortex
- Intel x86
- ...



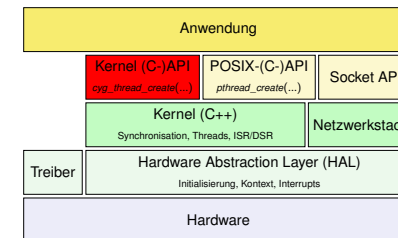
- Abstrahiert CPU- und plattformspezifische Eigenschaften
 - Kontextwechsel
 - Interruptverwaltung
 - CPU-Erkennung, Startup
 - Zeitgeber, I/O-Registerzugriffe



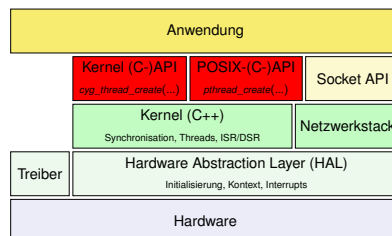
- Implementiert in C++
- Feingranular konfigurierbar
 - Verschiedene Schedulingstrategien (Bitmap/Multilevel Queue)
 - Zeitscheibenbasiert, präemptiv, prioritätenbasiert
- Verschiedene Synchronisationsstrategien
 - Mutexe, Semaphore, Bedingungsvariablen
 - Messages Boxes



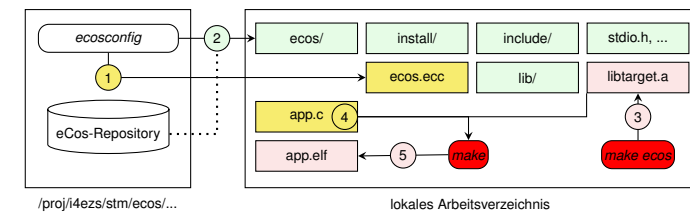
- Interrupt Service Routine (ISR)
 - Unverzügliche Ausführung
 - Asynchron
 - Kann DSR anfordern
- Deferred Service Routine (DSR)
 - Verzögerte Ausführung (beim Verlassen des Kernels)
 - Synchron



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



- 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

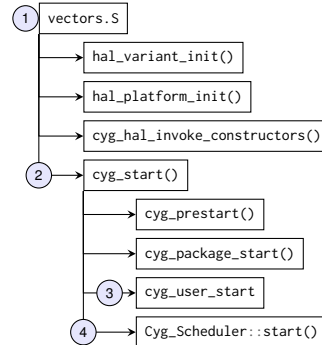
2 cyg_start():

- 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 --`

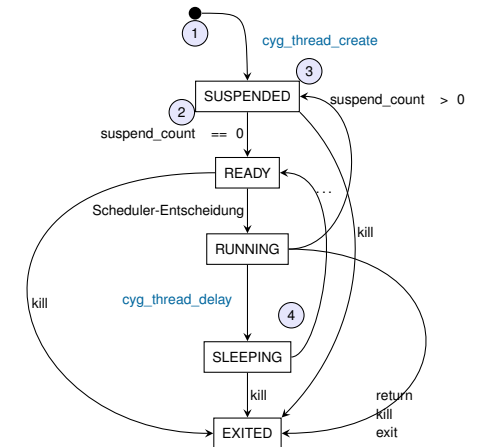
3 `cyg_thread_suspend` suspendiert

- `suspend_count++`

4 bereit

5 delay, mutex, semaphore wait

6 Threadterminierung



eCos-Beispielanwendung

Threads erzeugen

```
#include <cyg/kernel/kapi.h>
void cyg_thread_create
(
    cyg_addrword_t sched_info,
    cyg_thread_entry_t* entry,
    cyg_addrword_t entry_data,
    char* name,
    void* stack_base,
    cyg_ucount32 stack_size,
    cyg_handle_t* handle,
    cyg_thread* thread
);
```

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

Ausführliche Dokumentation

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



eCos-Beispielanwendung

Threads erzeugen

```
#include <cyg/kernel/kapi.h>
void cyg_thread_create
(
    cyg_addrword_t sched_info,
    cyg_thread_entry_t* entry,
    cyg_addrword_t entry_data,
    char* name,
    void* stack_base,
    cyg_ucount32 stack_size,
    cyg_handle_t* handle,
    cyg_thread* thread
);
```

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

Ausführliche Dokumentation

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



```
#include <cyg/kernel/kapi.h>
void cyg_thread_create
(
    cyg_addrword_t sched_info,
    cyg_thread_entry_t* entry,
    cyg_addrword_t entry_data,
    char* name,
    void* stack_base,
    cyg_uint32 stack_size,
    cyg_handle_t* handle,
    cyg_thread* thread
);
```

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

Ausführliche Dokumentation

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



```
#include <cyg/kernel/kapi.h>
void cyg_thread_create
(
    cyg_addrword_t sched_info,
    cyg_thread_entry_t* entry,
    cyg_addrword_t entry_data,
    char* name,
    void* stack_base,
    cyg_uint32 stack_size,
    cyg_handle_t* handle,
    cyg_thread* thread
);
```

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

Ausführliche Dokumentation

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



```
#include <cyg/kernel/kapi.h>
void cyg_thread_create
(
    cyg_addrword_t sched_info,
    cyg_thread_entry_t* entry,
    cyg_addrword_t entry_data,
    char* name,
    void* stack_base,
    cyg_uint32 stack_size,
    cyg_handle_t* handle,
    cyg_thread* thread
);
```

- Beliebiger Threadname
- Tipp: (gdb) info threads

Ausführliche Dokumentation

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



```
#include <cyg/kernel/kapi.h>
void cyg_thread_create
(
    cyg_addrword_t sched_info,
    cyg_thread_entry_t* entry,
    cyg_addrword_t entry_data,
    char* name,
    void* stack_base,
    cyg_uint32 stack_size,
    cyg_handle_t* handle,
    cyg_thread* thread
);
```

- 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>



```
#include <cyg/kernel/kapi.h>
void cyg_thread_create
(
    cyg_addrword_t sched_info,
    cyg_thread_entry_t* entry,
    cyg_addrword_t entry_data,
    char* name,
    void* stack_base,
    cyg_ucount32 stack_size,
    cyg_handle_t* handle,
    cyg_thread* thread
);
```

- Stackgröße in Bytes

Ausführliche Dokumentation

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



```
#include <cyg/kernel/kapi.h>
void cyg_thread_create
(
    cyg_addrword_t sched_info,
    cyg_thread_entry_t* entry,
    cyg_addrword_t entry_data,
    char* name,
    void* stack_base,
    cyg_ucount32 stack_size,
    cyg_handle_t* handle,
    cyg_thread* thread
);
```

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

Ausführliche Dokumentation

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



```
#include <cyg/kernel/kapi.h>
void cyg_thread_create
(
    cyg_addrword_t sched_info,
    cyg_thread_entry_t* entry,
    cyg_addrword_t entry_data,
    char* name,
    void* stack_base,
    cyg_ucount32 stack_size,
    cyg_handle_t* handle,
    cyg_thread* thread
);
```

- 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>



```
#include <cyg/kernel/kapi.h>
void cyg_thread_create
(
    cyg_addrword_t sched_info,
    cyg_thread_entry_t* entry,
    cyg_addrword_t entry_data,
    char* name,
    void* stack_base,
    cyg_ucount32 stack_size,
    cyg_handle_t* handle,
    cyg_thread* thread
);
```

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); }
```



eCos-Beispielanwendung

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)



Übersicht

- 1 Einführung in eCos
- 2 Entwicklungsumgebung
- 3 Debuggen mit GDB



Dokumentation zum EZS-Board

Wiki zum EZS-Board

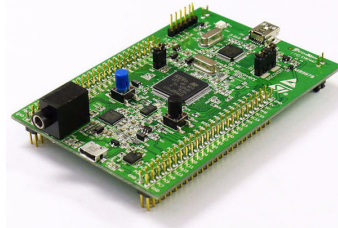
<https://gitlab.cs.fau.de/ezs/ezs-board/wikis/home>

- 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-Kaffeearte

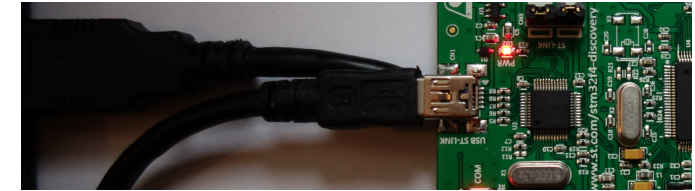


Entwicklungsplattform STM32F411

- 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*
 - ...



Flashen & Debuggen: Blackmagic Firmware



- 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



Flashen mittels Kommandozeile

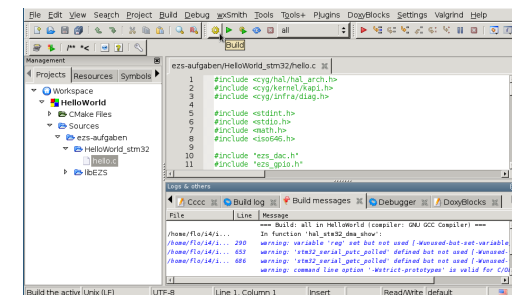
- Bashprompt: %, gdb-Prompt: >

```
% 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`



Editieren mit kate und codeblocks



- Anleitungen zum Editieren mit kate und codeblocks im Wiki
- Generieren der Projektdateien
 - `cmake -G"Kate - Unix Makefiles" ..`
 - `cmake -G"CodeBlocks - Unix Makefiles" ..`
- Weitere Makefile-Targets
 - `make flash`, `make gdb`, `make edit`, ...
 - Mehr dazu ~> Rechnerübung



Übersicht

1 Einführung in eCos

2 Entwicklungsumgebung

3 Debuggen mit GDB



FS, PW EZS (19.10.2017)
3 Debuggen mit GDB

24/29

gdb-Dashboard

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



FS, PW EZS (19.10.2017)
3 Debuggen mit GDB

25/29

```
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/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 {r3, 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>
0x0000458 ezs_counter_get+8 movs r1, #0
0x000045a ezs_counter_get+10 pop {r3, pc}
-- Stack
[8] from 0x0000452 in ezs_counter_get+2 at /home/noctux/work/ezs-aufgaben/ezs-aufgaben/HelloWorld/LibEZS/drivers/sta3274/ezs_counter.cpp:41
(no arguments)
[1] from 0x0000450 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.cpp:41
Locals
ezs_counter_get () at /home/noctux/work/ezs-aufgaben/ezs-aufgaben/HelloWorld/LibEZS/drivers/sta3274/ezs_counter.cpp:41
41 return timer_get_counter(TIM5);
Loading section .rom_vectors, size 0x8 lma 0x00000000
Loading section .ARM.exidx, size 0x8 lma 0x00000008
Loading section .text, size 0x424 lma 0x00000010
Loading section .rodata, size 0x1240 lma 0x00000938
Loading section .data, size 0x40 lma 0x00000e08
Start address 0x000010, load size 81504
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) <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 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 {r3, 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>
0x0000458 ezs_counter_get+8 movs r1, #0
0x000045a ezs_counter_get+10 pop {r3, pc}
-- Stack
[8] from 0x0000452 in ezs_counter_get+2 at /home/noctux/work/ezs-aufgaben/ezs-aufgaben/HelloWorld/LibEZS/drivers/sta3274/ezs_counter.cpp:41
(no arguments)
[1] from 0x0000450 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.cpp:41
Locals
ezs_counter_get () at /home/noctux/work/ezs-aufgaben/ezs-aufgaben/HelloWorld/LibEZS/drivers/sta3274/ezs_counter.cpp:41
41 return timer_get_counter(TIM5);
Loading section .rom_vectors, size 0x8 lma 0x00000000
Loading section .ARM.exidx, size 0x8 lma 0x00000008
Loading section .text, size 0x424 lma 0x00000010
Loading section .rodata, size 0x1240 lma 0x00000938
Loading section .data, size 0x40 lma 0x00000e08
Start address 0x000010, load size 81504
Transfer rate: 16 KB/sec, 918 bytes/write.
>>> break hello.c:40
Breakpoint 1 at 0x00000dec: file /home/noctux/work/ezs-aufgaben/ezs-aufgaben/HelloWorld/hello.c, line 40.
>>> []
```



FS, PW EZS (19.10.2017)
3 Debuggen mit GDB

26/29

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>



FS, PW EZS (19.10.2017)
3 Debuggen mit GDB

27/29

```
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("Hello Welt\n");
42
43 up = not up;
44 ezs_gpio_set(up);
45 }
-- Assembly
0x0000000 test_thread+20 ldr r8, [pc, #132] ; 0x000010c <test_thread+156>
0x0000002 test_thread+22 ldr r7, [pc, #108] ; (0x0000150 <test_thread+136>)
0x0000004 test_thread+24 ldr r6, [pc, #112] ; (0x000015c <test_thread+140>)
0x0000006 test_thread+26 mov r4, r4
0x0000008 test_thread+28 asrs r3, r4, #1
0x000000a test_thread+30 mov w r2, #1000
0x000000c test_thread+32 movs r3, #0
-- Stack
[8] from 0x0000000 in test_thread+28 at /home/noctux/work/ezs-aufgaben/ezs-aufgaben/HelloWorld/hello.c:40
arg = <optimized out>
[1] from 0x0000000 in Sys_HardwareThread::thread_entry+18 at /home/noctux/work/ezs-aufgaben/ezs-aufgaben/HelloWorld/hello.c:40
arg thread = 0x0000070 <threaddata>
[1]
-- Threads
[1] id 0 from 0x0000000 in test_thread+28 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 0x00000dec 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 0x0000170 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“:
`% 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>

