DIY – Individual Prototyping and Systems Engineering

Peter Ulbrich

Lehrstuhl für Verteilte Systeme und Betriebssysteme

Friedrich-Alexander-Universität Erlangen-Nürnberg

`https://www4.cs.fau.de`

20. Mai 2019

Übersicht

eCos

Embedded **C**onfigurable **O**perating **S**ystem

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.sourceware.org`

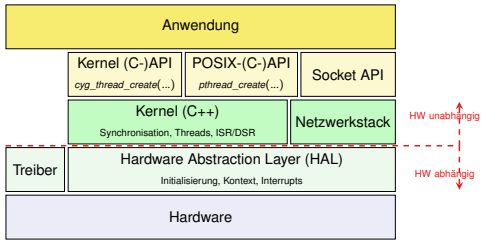
Unterstützte Plattformen

`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*
- ...

eCos-Systemarchitektur

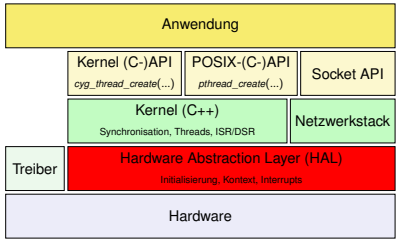
Überblick



eCos-Systemarchitektur

Hardware Abstraction Layer

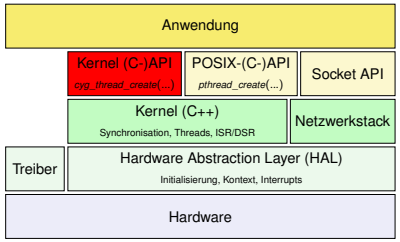
- Abstrahiert CPU- und plattformspezifische Eigenschaften
  - Kontextwechsel
  - Interruptverwaltung
  - CPU-Erkennung, Startup
  - Zeitgeber, I/O-Registerzugriffe



eCos-Systemarchitektur

Kernel

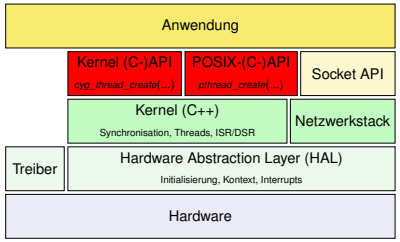
- Implementiert in C++
- Feingranular konfigurierbar
  - Verschiedene Schedulingstrategien (Bitmap/Multilevel Queue)
  - Zeitscheibenbasiert, präemptiv, prioritätenbasiert
- Verschiedene Synchronisationsstrategien
  - Mutexe, Semaphore, Bedingungsvariablen
  - Messages Boxes



eCos-Systemarchitektur

Application Programming Interface (API)

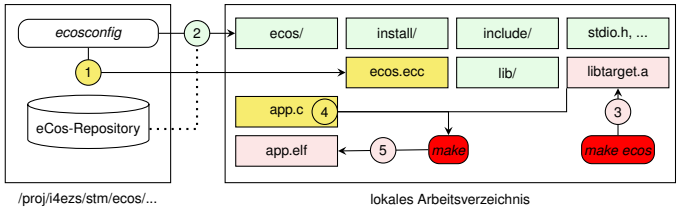
- Kernel API
  - vollständige C-Schnittstelle
  - siehe Dokumentation[1]
- (Optionale) POSIX-Kompatibilitätsschicht
  - Scheduling-Konfiguration, *pthread_∗*
  - Timer, Semaphore, Message Queues, Signale, ...



[1] `http://ecos.sourceware.org/docs-2.0/ref/ecos-ref.html`

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

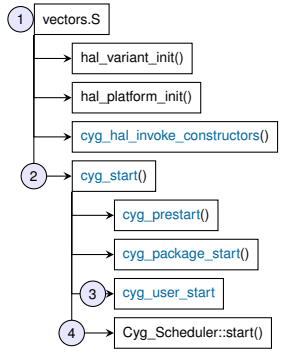


Wichtig!

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!

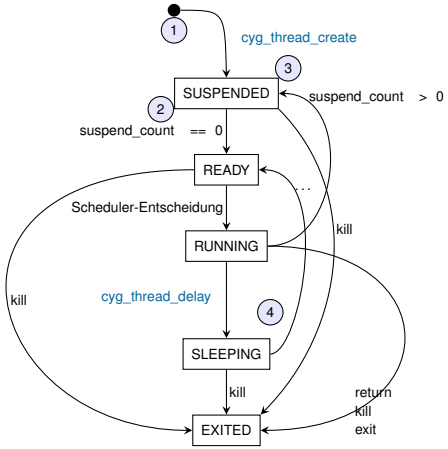
In allen Übungsaufgaben muss man `cyg_user_start()` implementieren und dort alle Threads anlegen.

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. bereit
5. delay, mutex, semaphore wait
6. Threadterminierung



eCos-Beispielanwendung

Threads erzeugen

```
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`

eCos-Beispielanwendung

Threads erzeugen

```
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`

eCos-Beispielanwendung

Threads erzeugen

```
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`

pu DIY (20. Mai 2019) – Kapitel V Embedded Entwicklung I 12/32
1 Einführung in eCos

eCos-Beispielanwendung

Threads erzeugen

```
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`

pu DIY (20. Mai 2019) – Kapitel V Embedded Entwicklung I 12/32
1 Einführung in eCos

eCos-Beispielanwendung

Threads erzeugen

```
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`

pu DIY (20. Mai 2019) – Kapitel V Embedded Entwicklung I 12/32
1 Einführung in eCos

eCos-Beispielanwendung

Threads erzeugen

```
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`

eCos-Beispielanwendung

Threads erzeugen

```
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`

pu DIY (20. Mai 2019) – Kapitel V Embedded Entwicklung I 12/32
1 Einführung in eCos

eCos-Beispielanwendung

Threads erzeugen

```
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`

pu DIY (20. Mai 2019) – Kapitel V Embedded Entwicklung I 12/32
1 Einführung in eCos

eCos-Beispielanwendung

Threads erzeugen

```
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`

pu DIY (20. Mai 2019) – Kapitel V Embedded Entwicklung I 12/32
1 Einführung in eCos

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
);
```

Ausführliche Dokumentation

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

eCos-Beispielanwendung

```
#include <cyg/hal/hal_arch.h>
#include <cyg/kernel/kapi.h>
#include <stdio.h>
#define MY_PRIORITY    11
#define STACKSIZE      (CYGNUM_HAL_STACK_SIZE_MINIMUM+4096)
static cyg_uint8     my_stack[STACKSIZE];
static cyg_handle_t my_handle;
static cyg_thread   my_thread;

static void my_entry(cyg_addrword_t data) {
  int message = (int) data;
  ezs_printf("Beginning execution: thread data is %d\n", message);
  for (;;) {
    ezs_printf("Hello World!\n"); // \n flushes output
    ezs_delay_us(1000000); // Delay for 1000000 * 1us = 1 second
  }
}
void cyg_user_start(void) {
    ezs_printf("Entering cyg_user_start() function\n");
    cyg_thread_create(MY_PRIORITY, &my_entry, 0, "thread 1",
                      my_stack, STACKSIZE, &my_handle, &my_thread);
    cyg_thread_resume(my_handle); }
```

Gliederung

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 build[2]
5. Makefiles erzeugen: `cmake ..` (*cmake PUNKT PUNKT*)
6. Alles kompilieren: `make`
7. Flashen, ausführen, debuggen: `make flash`

⇝ Flasher & Debugger: `make gdb` (später ausführlicher)

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

Dokumentation zum EZS-Board

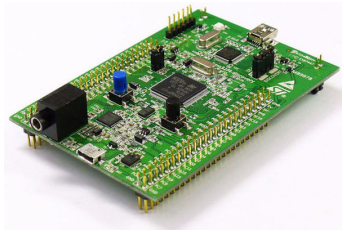
Wiki zum EZS-Board

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

- Nicht unterstützte Plattform
- ☞ Dokumentation im Wiki
- Erweiterung durch *alle Teilnehmer an EZS/VEZS/DIY*
  - gitlab account notwendig
- Besondere Aufgaben
  - Anleitung „EZS-Board unter Windows“
  - Anleitung „EZS-Board unter macOS“
  - ☞ Gutscheine für I4-Kaffeekarte

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 Firmware[3] bereits ausgestattet

[3] `https://github.com/blacksphere/blackmagic`

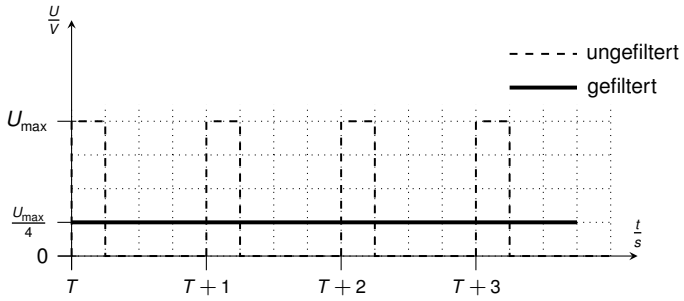
Flashen mittels Kommandozeile

- Bashprompt: %, gdb-Prompt: >

```
% cd <HelloDIY>
% 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
```

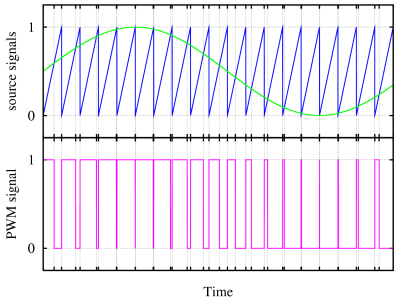
- `gdb -nh`: Verhindert Ausführung der Befehle aus `~/.gdbinit`
- gdb-Befehle in Datei `flash.gdb` zusammenfassen und starten mittels:
  `arm-none-eabi-gdb -batch -x flash.gdb -nh app.elf`

Digital-Analog Umwandlung & Pulsweitenmodulation I



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

Digital-Analog Umwandlung & Pulsweitenmodulation II



Verfahren zur **Signalerzeugung**

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

Tiefpassfilter

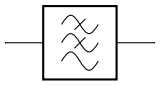


Abbildung: Schaltbild RC-Glied

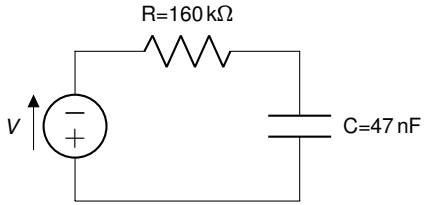
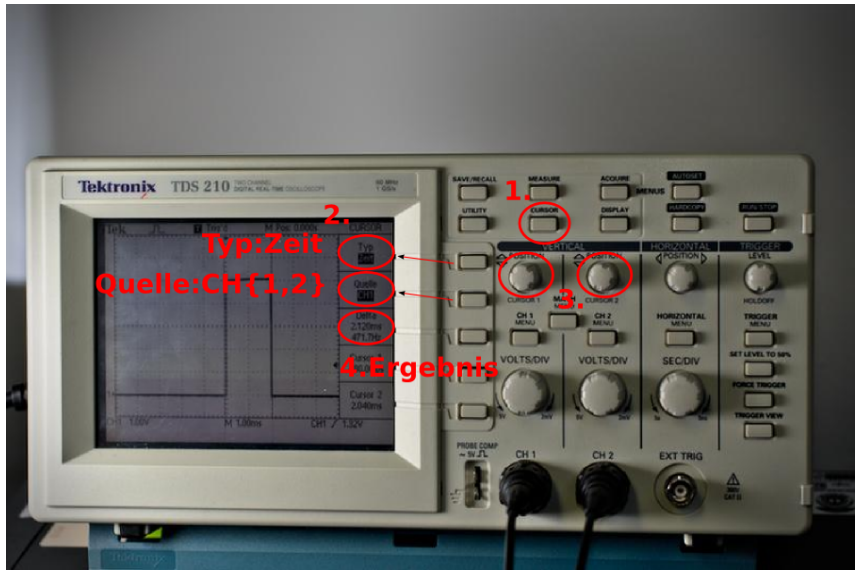


Abbildung: Schaltung RC-Filter auf EZS-Board

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

Cursor



Umgang mit Zeit in eCos

- 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;
```

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

Gliederung

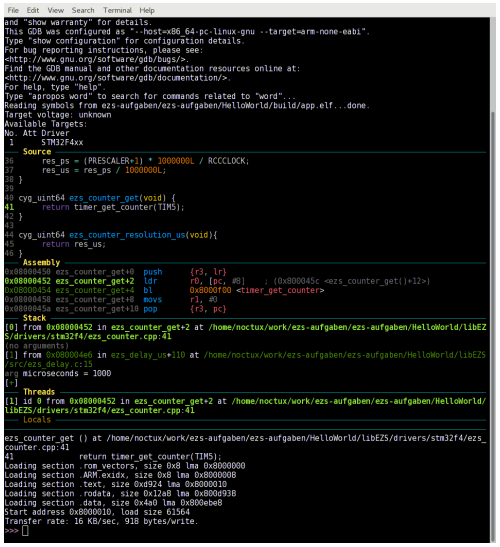
gdb-Dashboard

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 Varibalen

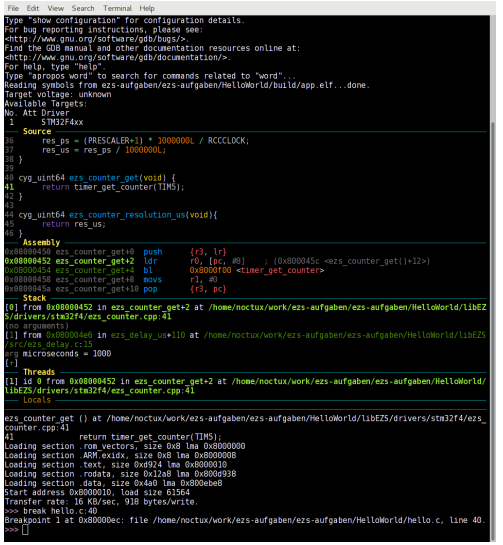


gdb Kommandos – I

Befehle haben Langformen (`break`) und Kurzformen (`b`)

Wichtige Befehle

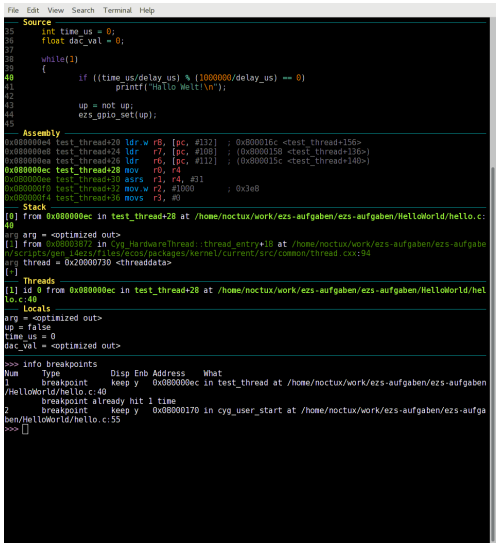
- Breakpoint setzten:
  `» b(reak) cyg_user_start`
- Einzelschritt (Funktionen betreten):
  `» s(tep)`
- Einzelschritt (Funktionen nicht betreten):
  `» n(ext)`
- Programm fortsetzten:
  `» c(ontinue)`
- Bis zum Ende der Funktion ausführen:
  `» fin(ish)`
- Funktion anzeigen:
  `» l(ist) <funktionsname>`
- gdb schließen:
  `» q(uit) (oder Strg+D)`
- Neu Flashen:
  `» l(oad)`



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>`



Weiterführende Informationen

- 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`

Gliederung

Hilfe bei Auswahl von Lizenzen

- Nützliche Links
  - `https://choosealicense.com/`
  - `https://help.github.com/articles/licensing-a-repository/`
  - `https://tldrlegal.com/`
- Welche Lizenzen haben die verwendeten Komponenten?
  - Board-Support von ST
  - `libdiy`?

☞ Sorgfalt bei der Auswahl der Lizenz