Echtzeitsysteme - Übungen

Entwicklungsumgebung

Sommersemester 2024

Eva Dengler Peter Wägemann

Friedrich-Alexander-Universität Erlangen-Nürnberg (FAU)
Lehrstuhl Informatik 4 (Systemsoftware)
https://sys.cs.fau.de

Lehrstuhl für Informatik 4
Systemsoftware

Friedrich-Alexander-Universität
Technische Fakultät

Übersicht

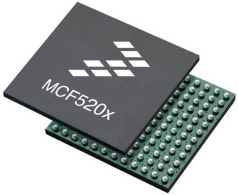
Einführung in eCos

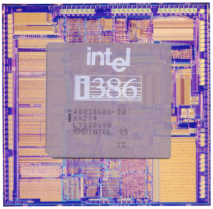
Entwicklungsumgebung

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

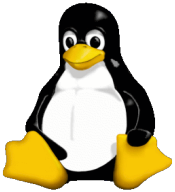
Debuggen mit GDB

Prozessorvielfalt in der Echtzeitwelt





Noch mehr Betriebssysteme



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

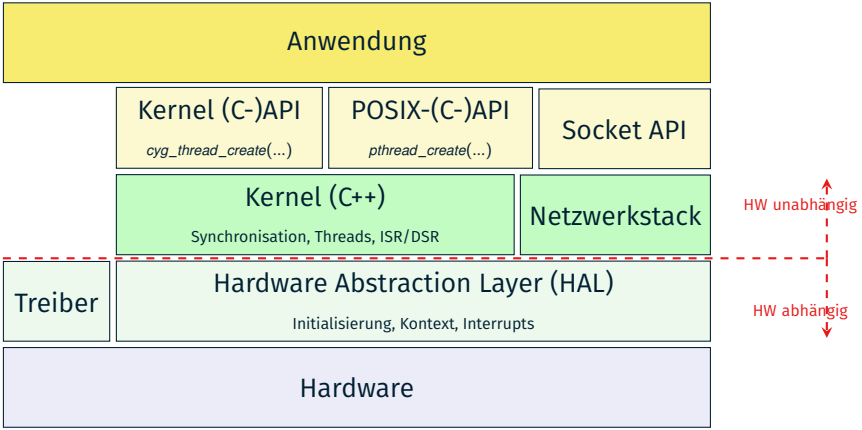
- Fujitsu SPARClite
- Matsushita MN10300
- Motorola PowerPC
- Advanced RISC Machines (ARM)
- Toshiba TX39
- Infineon TriCore
- Hitachi SH3
- NEC VR4300
- MB8683X

⇒ *ARM Cortex*

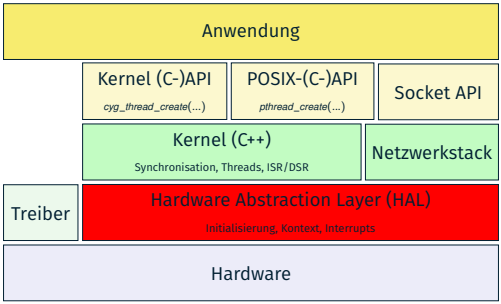
⇒ *Intel x86*

- ...

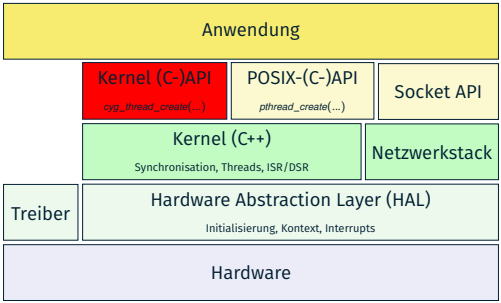
eCos-Systemarchitektur



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

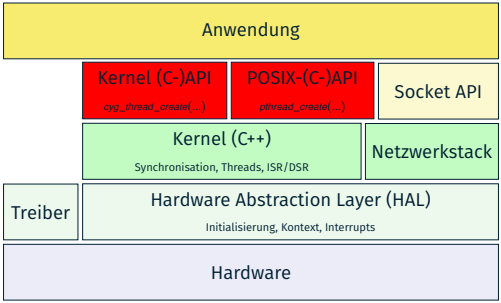


eCos-Systemarchitektur



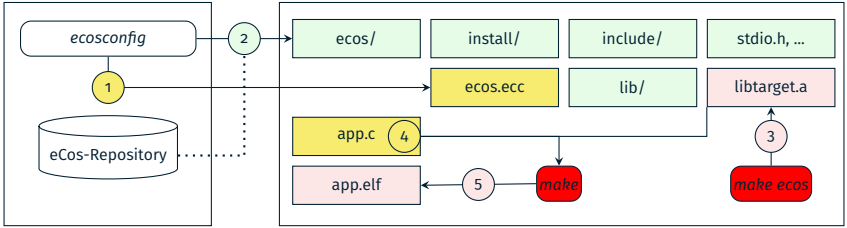
- Implementiert in C++
- Feingranular konfigurierbar
  - Verschiedene Schedulingstrategien (Bitmap/Multilevel Queue)
  - Zeitscheibenbasiert, präemptiv, prioritätenbasiert
- Verschiedene Synchronisationsstrategien
  - Mutexe, Semaphore, Bedingungsvariablen
  - Messages Boxes

- Kernel API
  - vollständige C-Schnittstelle
  - Dokumentation `http://ecos.sourceware.org/docs-2.0/ref/ecos-ref.html`
- (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



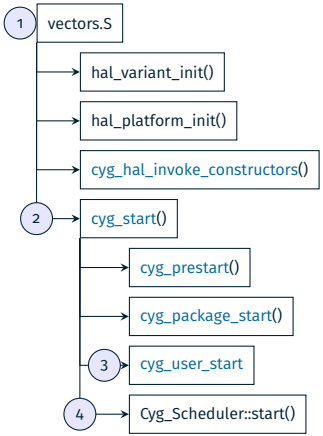
/proj/i4ezs/stm/ecos/...

lokales Arbeitsverzeichnis

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



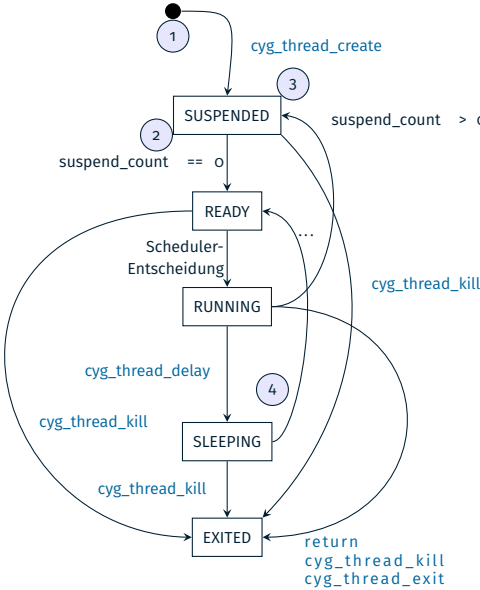
Wichtig!

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.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 );
```

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

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 );
```

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

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 );
```

- 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)`
  - `cyg_thread_set_priority(handle)`
  - `cyg_thread_kill(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

```c
#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);  }
```

Übersicht

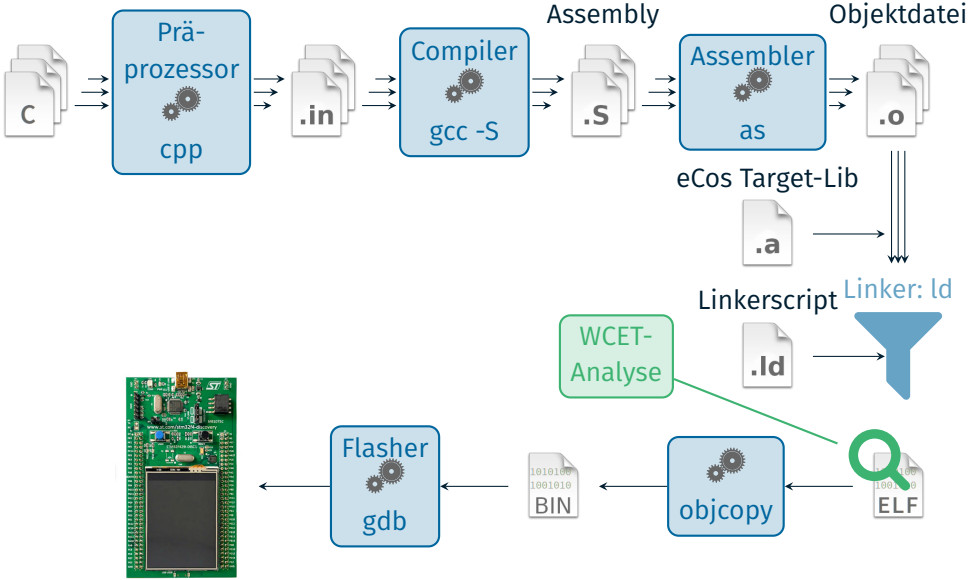
Einführung in eCos

Entwicklungsumgebung

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

Debuggen mit GDB

EZS-Toolchain



EZS-Entwicklungsumgebung

Zu jeder Übungsaufgabe wird eine eCos-Konfiguration bereitgestellt.

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[1]
5. Makefiles erzeugen: `cmake ..` (*cmake PUNKT PUNKT*)
6. Alles kompilieren: `make`
7. Flashen, ausführen: `make flash` ⇝ Debugging: `make gdb` (später ausführlicher)

[1] https://gitlab.kitware.com/cmake/community/wikis/FAQ#out-of-source-build-trees

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

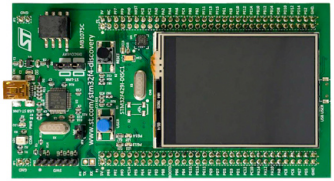
  3-Achsen Gyroskop

  Beschleunigungssensor

  Audio Sensor

  integrierte Debugging-Schnittstelle

  ...



Flashen & Debuggen: Blackmagic Firmware

- Ausleihbare Boards sind mit Blackmagic Firmware[2] 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 $\rightsquigarrow$ `/dev/serial/by-id/*Black_Magic*`

$\Rightarrow$ Für EZS: `/tmp/$USER-ezs-serial`

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

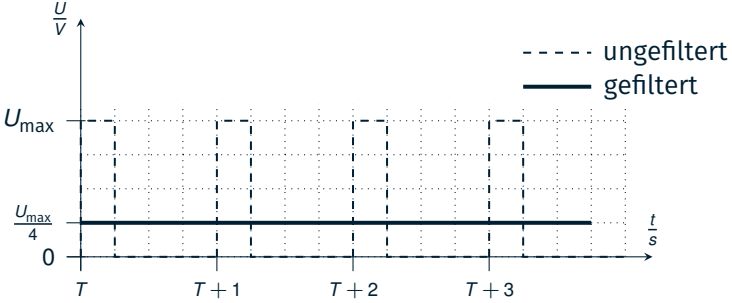
Flashen mittels Kommandozeile

- 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
```

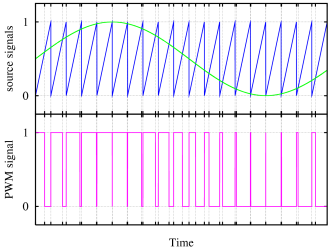
- `gdb -nh`: Verhindert Ausführung der Befehle aus `~/.gdbinit`
- gdb-Befehle in Datei `flash.gdb` zusammenfassen und starten mittels:
  `arm-none-eabi-gdb -batch -x flash.gdb -nh app.elf`
- bereits implementiert in Target `make flash`

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 $\rightsquigarrow$ Digital-Analog-Wandlung
- libEZS: `void ezs_dac_write(uint8_t)` (zusätzlich *echter* DAC auf Board)
  $\rightsquigarrow$ Wertebereich 0-255

Tiefpassfilter

Abbildung 1: Schaltbild RC-Glied

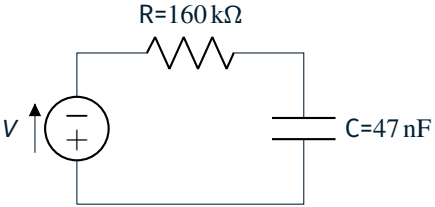


Abbildung 2: Schaltung RC-Filter auf Filterbaustein

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

Tiefpassfilter anschließen

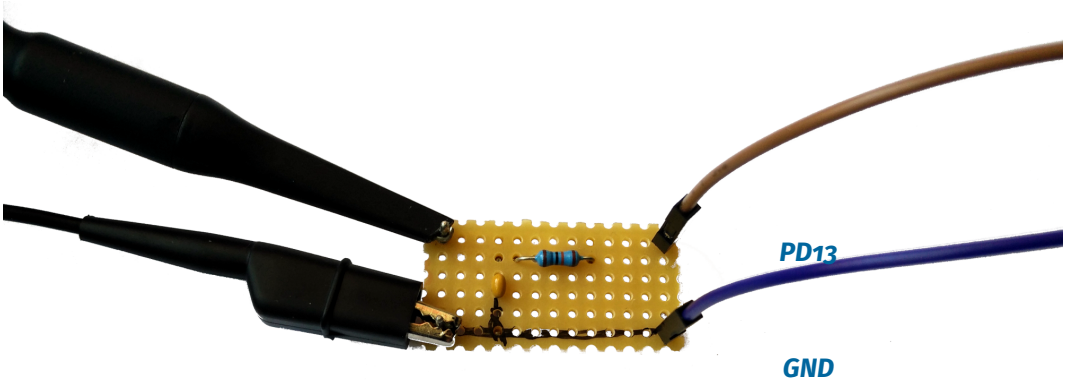
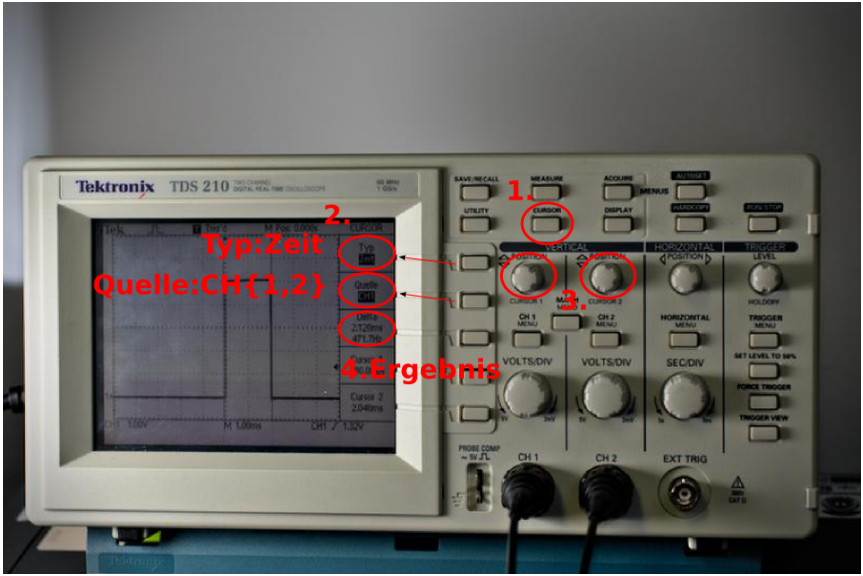


Abbildung 3: Filterbaustein anschließen, *schwarzer Strich markiert Masse*

Cursor


1.
2.
Typ:Zeit
Quelle:CH{1,2}
4.Ergebnis
Tektronix
TDS 210
CURSOR
VERTICAL
HORIZONTAL
TRIGGER
VOLTS/DIV
SEC/DIV
CH 1
CH 2
EXT TRIG

Übersicht

Einführung in eCos

Entwicklungsumgebung

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

Debuggen mit GDB

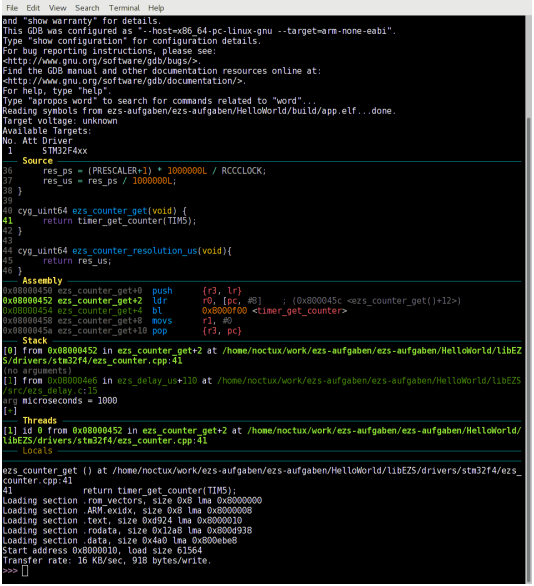
gdb-Dashboard

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



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)`

```
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 / RCCCLOCK;
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    {r3, lr}
0x08000452 ezs_counter_get+2  ldr     r0, [pc, #8]    ; (0x800045c <ezs_counter_get()+12>)
0x08000454 ezs_counter_get+4  bl      0x8000f00 <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/libEZ
S/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
[+]
   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 0xd924 lma 0x8000010
Loading section .rodata, size 0x12a8 lma 0x8004938
Loading section .data, size 0x4a0 lma 0x800ebe8
Start address 0x8000010, load size 61564
Transfer rate: 16 KB/sec, 918 bytes/write.
>>> break hello.c:40
Breakpoint 1 at 0x8000dec: file /home/noctux/work/ezs-aufgaben/ezs-aufgaben/HelloWorld/hello.c, line 40.
>>>
```

gdb Kommandos – II

Wichtige Befehle (Fortsetzung)

- Backtrace (Aufruf-Stack) anzeigen:

  »> `b(ack)t(race)`

- Dashboard neu zeichnen:

  »> `dashboard`

- Breakpoints anzeigen:

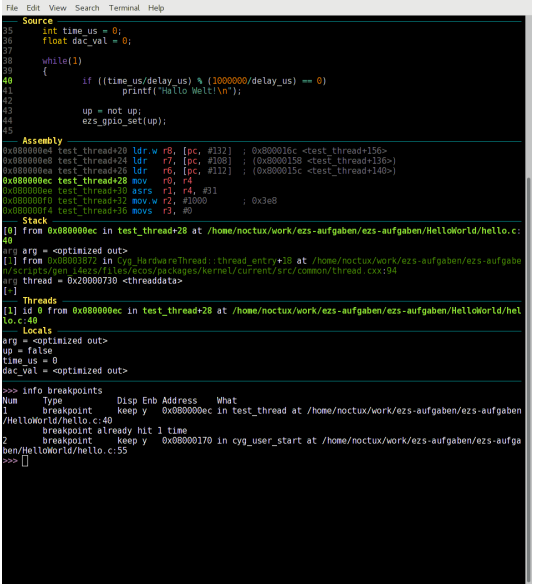
  »> `info breakpoints`

- Breakpoint löschen:

  »> `delete <nummer>`

- Variable anzeigen:

  »> `p(rint) <variablenname>`



Weiterführende Informationen

- gdb-Dashboard benötigt einen gdb mit python-Bindings
- gdb „abschießen“:[3]

  ```
  % 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://sys.cs.fau.de/extern/lehre/ws23/bs/uebung/seminar/gdb.pdf`

[3] `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 interaktive 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.