

DIY – Individual Prototyping and Systems Engineering

Embedded Entwicklung

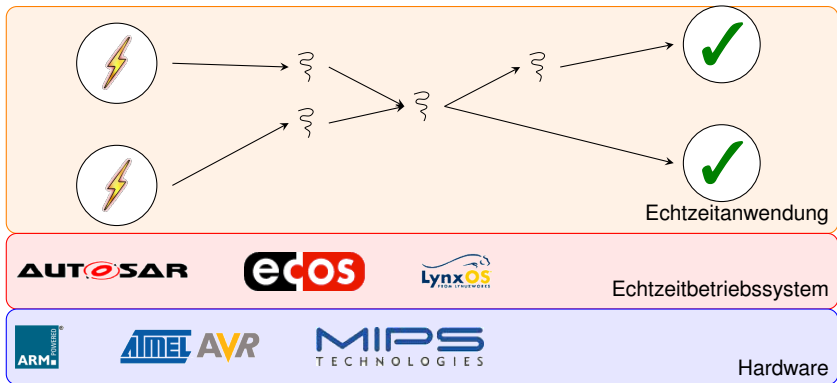
Peter Wägemann

Lehrstuhl für Verteilte Systeme und Betriebssysteme
Friedrich-Alexander-Universität Erlangen-Nürnberg

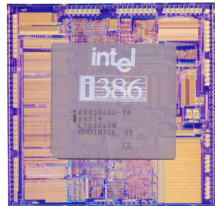
<https://www4.cs.fau.de>

30. April 2018



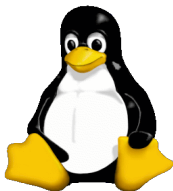


Prozessorvielfalt in der Echtzeitwelt



free **RTOS**

μClinux



eCos

QNX



μC/OS-II™
The Real-Time Kernel

HIGH TEC



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>

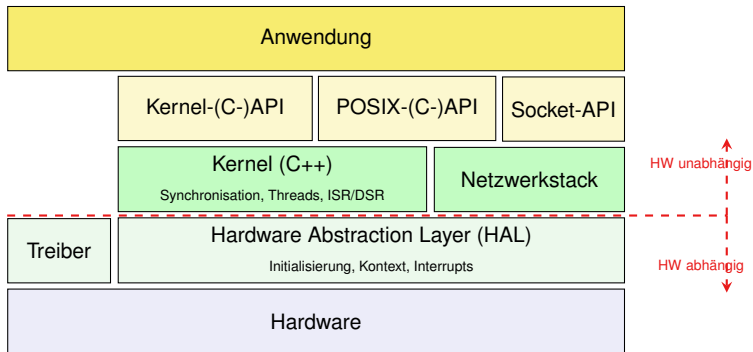


Unterstützte Plattformen

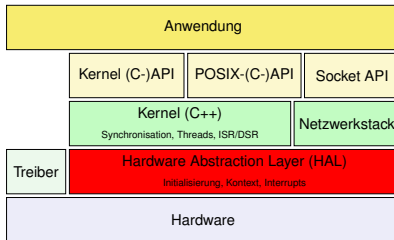
<http://www.ecoscentric.com/ecos/examples.shtml>

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

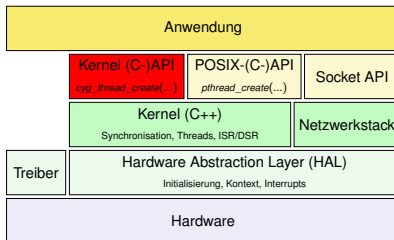




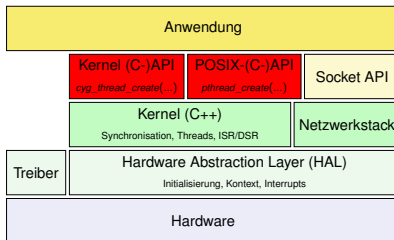
- Abstrahiert CPU- und plattformspezifische Eigenschaften
 - Kontextwechsel
 - Interruptverwaltung
 - CPU-Erkennung, Startup
 - Zeitgeber, I/O-Registerzugriffe



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

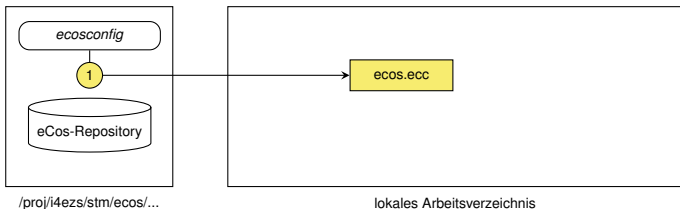


- Kernel API
 - vollständige C-Schnittstelle
 - siehe Dokumentation¹
- (Optionale) POSIX-Kompatibilitätsschicht
 - Scheduling-Konfiguration, *pthread*_*
 - Timer, Semaphore, Message Queues, Signale, ...

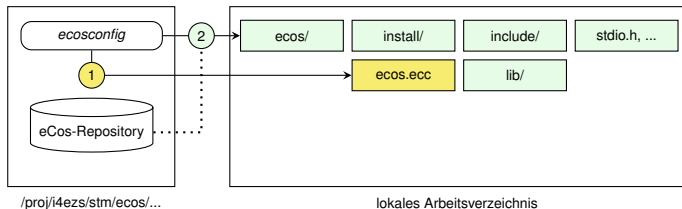


¹<http://ecos.sourceforge.org/docs-2.0/ref/ecos-ref.html>

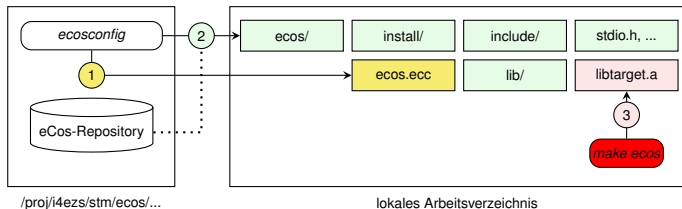
1 Erstellen einer Konfiguration (configtool/ecosconfig)



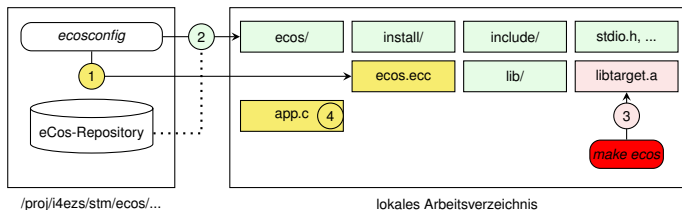
- 1 Erstellen einer Konfiguration (configtool/ecosconfig)
- 2 Kopieren ausgewählter Komponenten (configtool/ecosconfig)



- 1 Erstellen einer Konfiguration (configtool/ecosconfig)
- 2 Kopieren ausgewählter Komponenten (configtool/ecosconfig)
- 3 Erstellen einer Betriebssystem**bibliothek**

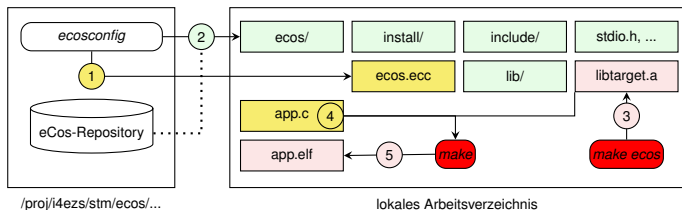


- 1 Erstellen einer Konfiguration (configtool/ecosconfig)
- 2 Kopieren ausgewählter Komponenten (configtool/ecosconfig)
- 3 Erstellen einer Betriebssystem**bibliothek**
- 4 Entwicklung der eigentlichen Anwendung



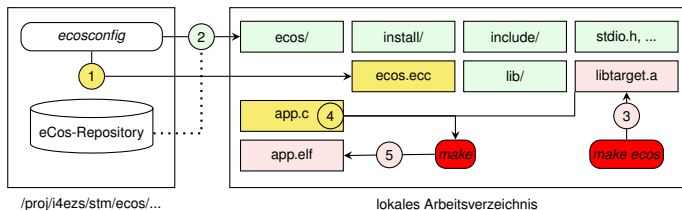
eCos-Entwicklungszyklus

- 1 Erstellen einer Konfiguration (configtool/ecosconfig)
- 2 Kopieren ausgewählter Komponenten (configtool/ecosconfig)
- 3 Erstellen einer Betriebssystem**bibliothek**
- 4 Entwicklung der eigentlichen Anwendung
- 5 Kompilieren des Gesamtsystems



eCos-Entwicklungszyklus

- 1 Erstellen einer Konfiguration (configtool/ecosconfig)
- 2 Kopieren ausgewählter Komponenten (configtool/ecosconfig)
- 3 Erstellen einer Betriebssystem**bibliothek**
- 4 Entwicklung der eigentlichen Anwendung
- 5 Kompilieren des Gesamtsystems



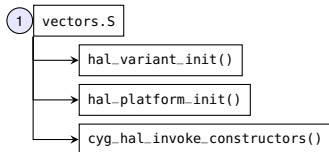
Wichtig!

Wir geben eine grundlegende Konfiguration vor!



1 vectors.S

- Hardwareinitialisierung
- Globale Konstruktoren

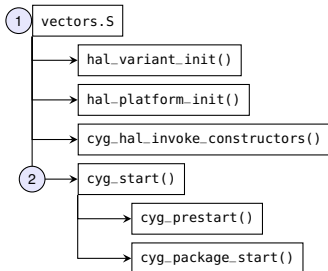


1 vectors.S

- Hardwareinitialisierung
- Globale Konstruktoren

2 cyg_start():

- Hardwareunabhängige Vorbereitungen



1 vectors.S

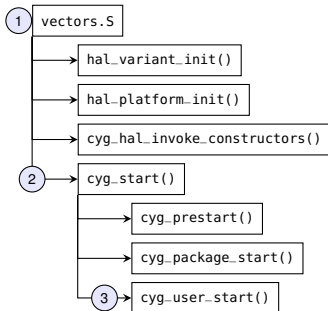
- Hardwareinitialisierung
- Globale Konstruktoren

2 cyg_start():

- Hardwareunabhängige Vorbereitungen

3 cyg_user_start():

- Einsprungpunkt für Anwendungscode!
- Erzeugen von Threads



1 vectors.S

- Hardwareinitialisierung
- Globale Konstruktoren

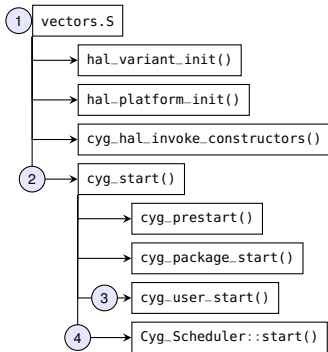
2 cyg_start():

- Hardwareunabhängige Vorbereitungen

3 cyg_user_start():

- Einsprungpunkt für Anwendungscode!
- Erzeugen von Threads

4 Starten des Schedulers



1 vectors.S

- Hardwareinitialisierung
- Globale Konstruktoren

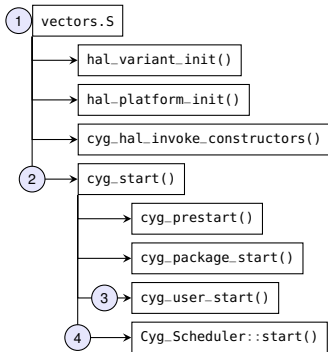
2 cyg_start():

- Hardwareunabhängige Vorbereitungen

3 cyg_user_start():

- Einsprungpunkt für Anwendungscode!
- Erzeugen von Threads

4 Starten des Schedulers



Wichtig!

cyg_user_start() muss zurückkehren!



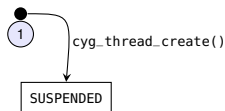
- 1 Einführung in eCos
- 2 eCos-Threads**
 - Zeit in eCos
- 3 Interruptbehandlung
 - Einschub: Schlüsselwort volatile
 - ISR & DSR
 - eCos-Unterbrechungsbehandlung
- 4 Periodische Ausführung – eCos Alarme
- 5 Ereignisse in eCos
 - Events
 - Mailbox
- 6 Zugriffskontrolle in eCos
- 7 Entwicklungsumgebung
- 8 Debuggen mit GDB



- 1 Thread wird im Zustand *suspended* erzeugt.

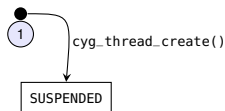
- `suspend_count = 1`

Threadzustände und Übergänge



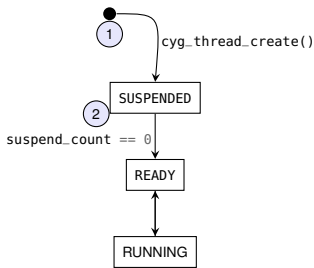
- 1 Thread wird im Zustand *suspended* erzeugt.
 - `suspend_count = 1`
- 2 `cyg_thread_resume()` aktiviert
 - `suspend_count --`

Threadzustände und Übergänge



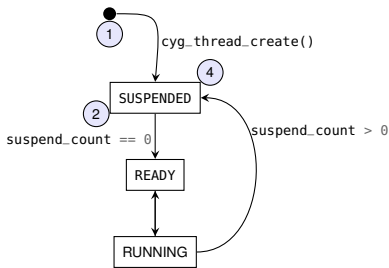
- 1 Thread wird im Zustand *suspended* erzeugt.
 - `suspend_count = 1`
- 2 `cyg_thread_resume()` aktiviert
 - `suspend_count --`
- 3 bereit \leftrightarrow laufend

Threadzustände und Übergänge



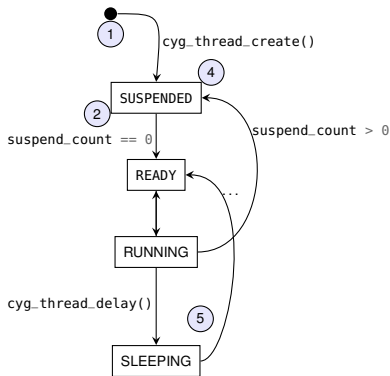
- 1 Thread wird im Zustand *suspended* erzeugt.
 - `suspend_count = 1`
- 2 `cyg_thread_resume()` aktiviert
 - `suspend_count --`
- 3 bereit \leftrightarrow laufend
- 4 `cyg_thread_suspend()` suspendiert
 - `suspend_count ++`

Threadzustände und Übergänge

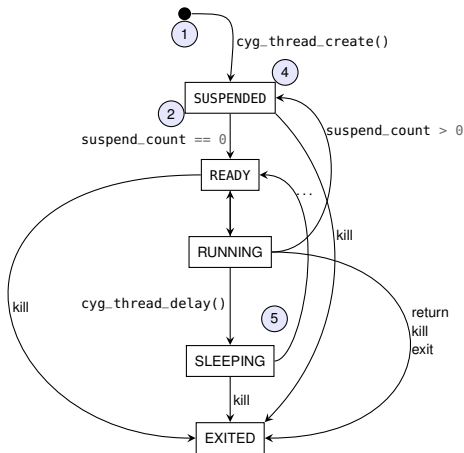


- 1 Thread wird im Zustand *suspended* erzeugt.
 - `suspend_count = 1`
- 2 `cyg_thread_resume()` aktiviert
 - `suspend_count --`
- 3 bereit \leftrightarrow laufend
- 4 `cyg_thread_suspend()` suspendiert
 - `suspend_count ++`
- 5 `delay`, `mutex`, `semaphore` `wait`

Threadzustände und Übergänge



- 1 Thread wird im Zustand *suspended* erzeugt.
 - `suspend_count = 1`
- 2 `cyg_thread_resume()` aktiviert
 - `suspend_count --`
- 3 bereit \leftrightarrow laufend
- 4 `cyg_thread_suspend()` suspendiert
 - `suspend_count++`
- 5 delay, mutex, semaphore wait
- 6 Threadterminierung



```
#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



```
#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



```
#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
);
```

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



```
#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
);
```

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



```
#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
);
```

- Beliebiger Threadname
- Tipp: (gdb) info threads



```
#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
);
```

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



```
#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



```
#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)



```
#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



```
#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>



```
#include <cyg/hal/hal_arch.h>
#include <cyg/kernel/kapi.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
    }
}

void cyg_user_start(void) {
    cyg_thread_create(MY_PRIORITY, &my_entry, 0, "thread 1",
        my_stack, STACKSIZE, &my_handle, &my_thread);
    cyg_thread_resume(my_handle);
}
```



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?*



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:

```
typedef struct {  
    cyg_uint32 dividend;  
    cyg_uint32 divisor;  
} cyg_resolution_t;
```

- $\frac{\text{dividend}}{\text{divisor}}$ → Zeit in ns, die ein Tick dauert (beispielsweise 1.000.000 ns)
- *Warum Aufteilung in Dividend & Divisor?*
- Umrechnung: z.B. `ms_to_cyg_ticks(uint32_t ms)`
- Vorsicht: Ganzzahl-Division, Wertebereichen, Fließkomma-Genauigkeit



- 1 Einführung in eCos
- 2 eCos-Threads
 - Zeit in eCos
- 3 Interruptbehandlung**
 - Einschub: Schlüsselwort volatile
 - ISR & DSR
 - eCos-Unterbrechungsbehandlung
- 4 Periodische Ausführung – eCos Alarme
- 5 Ereignisse in eCos
 - Events
 - Mailbox
- 6 Zugriffskontrolle in eCos
- 7 Entwicklungsumgebung
- 8 Debuggen mit GDB



Einschub: Schlüsselwort volatile

- Bei einem Interrupt wird `timer_event = 1` gesetzt
- Aktive Warteschleife wartet, bis `timer_event != 0`

```
static uint8_t timer_event = 0;
ISR (INT0_vect) { timer_event = 1; }

void main(void) {
    while(1) {
        while(timer_event == 0) { /* warte auf Timer-Event */
            /* bearbeite Timer-Event */
        }
    }
}
```



Einschub: Schlüsselwort volatile

- Bei einem Interrupt wird `timer_event = 1` gesetzt
- Aktive Warteschleife wartet, bis `timer_event != 0`
- Flag (scheinbar) in Schleife nicht verändert \leadsto Compiler-Optimierung
 - `timer_event` wird einmalig vor der Warteschleife in Register geladen
 - ☞ Endlosschleife

```
static uint8_t timer_event = 0;
ISR (INT0_vect) { timer_event = 1; }

void main(void) {
    while(1) {
        while(timer_event == 0) { /* warte auf Timer-Event */
            /* bearbeite Timer-Event */
        }
    }
}
```



- Bei einem Interrupt wird `timer_event = 1` gesetzt
- Aktive Warteschleife wartet, bis `timer_event != 0`
- Flag (scheinbar) in Schleife nicht verändert \leadsto Compiler-Optimierung
 - `timer_event` wird einmalig vor der Warteschleife in Register geladen
 - ☞ Endlosschleife
- `volatile` erzwingt das **Laden bei jedem Lesezugriff**

```
volatile static uint8_t timer_event = 0;
ISR (INT0_vect) { timer_event = 1; }

void main(void) {
    while(1) {
        while(timer_event == 0) { /* warte auf Timer-Event */
            /* bearbeite Timer-Event */
        }
    }
}
```



- Tastendruckzähler: Zählt mittels Variable `zaehler`
 - Inkrementierung in der Unterbrechungsbehandlung
 - Dekrementierung im Hauptprogramm zum Start der Verarbeitung

Hauptprogramm H

```
; volatile uint8_t zaehler;  
; C-Anweisung: zaehler--;  
lds r24, zaehler  
dec r24  
sts zaehler, r24
```

Interruptbehandlung I

```
; C-Anweisung: zaehler++  
lds r25, zaehler  
inc r25  
sts zaehler, r25
```

Zeile	zaehler	r24	r25
-	5		



- Tastendruckzähler: Zählt mittels Variable `zaehler`
 - Inkrementierung in der Unterbrechungsbehandlung
 - Dekrementierung im Hauptprogramm zum Start der Verarbeitung

Hauptprogramm H

```
; volatile uint8_t zaehler;  
; C-Anweisung: zaehler--;  
lds r24, zaehler  
dec r24  
sts zaehler, r24
```

Interruptbehandlung I

```
; C-Anweisung: zaehler++  
lds r25, zaehler  
inc r25  
sts zaehler, r25
```

Zeile	zaehler	r24	r25
-	5		
3 H	5	5	-



- Tastendruckzähler: Zählt mittels Variable `zaehler`
 - Inkrementierung in der Unterbrechungsbehandlung
 - Dekrementierung im Hauptprogramm zum Start der Verarbeitung

Hauptprogramm H

```
; volatile uint8_t zaehler;  
; C-Anweisung: zaehler--;  
lds r24, zaehler  
dec r24  
sts zaehler, r24
```

Interruptbehandlung I

```
; C-Anweisung: zaehler++  
lds r25, zaehler  
inc r25  
sts zaehler, r25
```

Zeile	zaehler	r24	r25
-	5		
3 H	5	5	-
4 H	5	4	-

- Tastendruckzähler: Zählt mittels Variable `zaehler`
 - Inkrementierung in der Unterbrechungsbehandlung
 - Dekrementierung im Hauptprogramm zum Start der Verarbeitung

Hauptprogramm H

```
; volatile uint8_t zaehler;  
; C-Anweisung: zaehler--;  
lds r24, zaehler  
dec r24  
sts zaehler, r24
```

Interruptbehandlung I

```
; C-Anweisung: zaehler++  
lds r25, zaehler  
inc r25  
sts zaehler, r25
```

Zeile	zaehler	r24	r25
-	5		
3 H	5	5	-
4 H	5	4	-
2 I	5	4	5

- Tastendruckzähler: Zählt mittels Variable `zaehler`
 - Inkrementierung in der Unterbrechungsbehandlung
 - Dekrementierung im Hauptprogramm zum Start der Verarbeitung

Hauptprogramm H

```
; volatile uint8_t zaehler;  
; C-Anweisung: zaehler--;  
lds r24, zaehler  
dec r24  
sts zaehler, r24
```

Interruptbehandlung I

```
; C-Anweisung: zaehler++  
lds r25, zaehler  
inc r25  
sts zaehler, r25
```

Zeile	zaehler	r24	r25
-	5		
3 H	5	5	-
4 H	5	4	-
2 I	5	4	5
3 I	5	4	6

- Tastendruckzähler: Zählt mittels Variable `zaehler`
 - Inkrementierung in der Unterbrechungsbehandlung
 - Dekrementierung im Hauptprogramm zum Start der Verarbeitung

Hauptprogramm H

```
; volatile uint8_t zaehler;  
; C-Anweisung: zaehler--;  
lds r24, zaehler  
dec r24  
sts zaehler, r24
```

Interruptbehandlung I

```
; C-Anweisung: zaehler++  
lds r25, zaehler  
inc r25  
sts zaehler, r25
```

Zeile	zaehler	r24	r25
-	5		
3 H	5	5	-
4 H	5	4	-
2 I	5	4	5
3 I	5	4	6
4 I	6	4	6

- Tastendruckzähler: Zählt mittels Variable `zaehler`
 - Inkrementierung in der Unterbrechungsbehandlung
 - Dekrementierung im Hauptprogramm zum Start der Verarbeitung

Hauptprogramm H

```
; volatile uint8_t zaehler;  
; C-Anweisung: zaehler--;  
lds r24, zaehler  
dec r24  
sts zaehler, r24
```

Interruptbehandlung I

```
; C-Anweisung: zaehler++  
lds r25, zaehler  
inc r25  
sts zaehler, r25
```

Zeile	zaehler	r24	r25
-	5		
3 H	5	5	-
4 H	5	4	-
2 I	5	4	5
3 I	5	4	6
4 I	6	4	6
5 H	4	4	-



Wie behandle ich einen Interrupt?

Interrupt-Service-Routinen-Ausführung

- Unverzüglich, *asynchron*
~> auch innerhalb von Kernelfunktionen!
- Innerhalb ISR *keine Systemaufrufe* erlaubt!
=> Anmelden einer Deferrable Service Routine (DSR)

Deferrable-Service-Routinen-Ausführung

- *Synchron* zum Scheduler
- Falls Scheduler nicht verriegelt: *Unverzüglich* nach ISR
- sonst: Beim *Verlassen* des Kerns

Synonym: *Prolog-Epilog-Schema* bzw. *top/bottom half*



Wie behandle ich einen Interrupt?

Anmeldung von ISR und DSR²

```
#include <cyg/kernel/kapi.h>
void cyg_interrupt_create
(
  cyg_vector_t vector ,
  cyg_priority_t priority ,
  cyg_addrword_t data ,
  cyg_ISR_t* isr ,
  cyg_DSR_t* dsr ,
  cyg_handle_t* handle ,
  cyg_interrupt* intr
);
```

■ Interruptvektornummer

~> Hardwarehandbuch

²<http://ecos.sourceware.org/docs-latest/ref/kernel-interrupts.html>

Wie behandle ich einen Interrupt?

Anmeldung von ISR und DSR²

```
#include <cyg/kernel/kapi.h>
void cyg_interrupt_create
(
    cyg_vector_t vector ,
    cyg_priority_t priority ,
    cyg_addrword_t data ,
    cyg_ISR_t* isr ,
    cyg_DSR_t* dsr ,
    cyg_handle_t* handle ,
    cyg_interrupt* intr
);
```

- Interruptpriorität
- für unterbrechbare Unterbrechungen (hardwareabhängig)

²<http://ecos.sourceware.org/docs-latest/ref/kernel-interrupts.html>



Wie behandle ich einen Interrupt?

Anmeldung von ISR und DSR²

```
#include <cyg/kernel/kapi.h>
void cyg_interrupt_create
(
    cyg_vector_t vector ,
    cyg_priority_t priority ,
    cyg_addrword_t data ,
    cyg_ISR_t* isr ,
    cyg_DSR_t* dsr ,
    cyg_handle_t* handle ,
    cyg_interrupt* intr
);
```

- Beliebiger Übergabeparameter für ISR/DSR

Für eCos:

build/ecos/install/include/cyg/hal/var_intr.h

²<http://ecos.sourceware.org/docs/latest/ref/kernel-interrupts.html>

Wie behandle ich einen Interrupt?

Anmeldung von ISR und DSR²

```
#include <cyg/kernel/kapi.h>
void cyg_interrupt_create
(
    cyg_vector_t vector ,
    cyg_priority_t priority ,
    cyg_addrword_t data ,
    cyg_ISR_t* isr ,
    cyg_DSR_t* dsr ,
    cyg_handle_t* handle ,
    cyg_interrupt* intr
);
```

- Funktionszeiger auf *ISR-Implementierung*

Signatur:

cyg_uint32 (*)(cyg_vector_t, cyg_addrword_t)

²<http://ecos.sourceware.org/docs/latest/ref/kernel-interrupts.html>

Wie behandle ich einen Interrupt?

Anmeldung von ISR und DSR²

```
#include <cyg/kernel/kapi.h>
void cyg_interrupt_create
(
    cyg_vector_t vector ,
    cyg_priority_t priority ,
    cyg_addrword_t data ,
    cyg_ISR_t* isr ,
    cyg_DSR_t* dsr ,
    cyg_handle_t* handle ,
    cyg_interrupt* intr
);
```

- Funktionszeiger auf *DSR-Implementierung*

Signatur:

```
cyg_uint32 (*)(cyg_vector_t, cyg_ucount32 count, cyg_addrword_t)
```

²<http://ecos.sourceware.org/docs/latest/ref/kernel-interrupts.html>



Wie behandle ich einen Interrupt?

Anmeldung von ISR und DSR²

```
#include <cyg/kernel/kapi.h>
void cyg_interrupt_create
(
    cyg_vector_t vector ,
    cyg_priority_t priority ,
    cyg_addrword_t data ,
    cyg_ISR_t* isr ,
    cyg_DSR_t* dsr ,
    cyg_handle_t* handle ,
    cyg_interrupt* intr
);
```

- Handle und Speicher für *Interruptobjekt*

²<http://ecos.sourceware.org/docs-latest/ref/kernel-interrupts.html>

Beispiel einer minimalen ISR

```
cyg_uint32 isr(cyg_vector_t vector, cyg_addrword_t data) {
    cyg_bool_t dsr_required = 0;
    ...
    cyg_interrupt_acknowledge(vector);
    if (dsr_required) {
        return CYG_ISR_CALL_DSR;
    } else {
        return CYG_ISR_HANDLED;
    }
}
```

1 Beliebiger ISR-Code



Beispiel einer minimalen ISR

```
cyg_uint32 isr(cyg_vector_t vector, cyg_addrword_t data) {  
    cyg_bool_t dsr_required = 0;  
    ...  
    cyg_interrupt_acknowledge(vector);  
    if (dsr_required) {  
        return CYG_ISR_CALL_DSR;  
    } else {  
        return CYG_ISR_HANDLED;  
    }  
}
```

- 1 Beliebiger ISR-Code
- 2 Bestätigung der Interruptbehandlung
Wozu ist das gut?



Beispiel einer minimalen ISR

```
cyg_uint32 isr(cyg_vector_t vector, cyg_addrword_t data) {
    cyg_bool_t dsr_required = 0;
    ...
    cyg_interrupt_acknowledge(vector);
    if (dsr_required) {
        return CYG_ISR_CALL_DSR;
    } else {
        return CYG_ISR_HANDLED;
    }
}
```

- 1 Beliebiger ISR-Code
- 2 Bestätigung der Interruptbehandlung
Wozu ist das gut?
- 3 Anforderung einer DSR
oder



Beispiel einer minimalen ISR

```
cyg_uint32 isr(cyg_vector_t vector, cyg_addrword_t data) {
    cyg_bool_t dsr_required = 0;
    ...
    cyg_interrupt_acknowledge(vector);
    if (dsr_required) {
        return CYG_ISR_CALL_DSR;
    } else {
        return CYG_ISR_HANDLED;
    }
}
```

- 1 Beliebiger ISR-Code
- 2 Bestätigung der Interruptbehandlung
Wozu ist das gut?
- 3 Anforderung einer DSR
oder
- 4 Rückkehr ohne DSR



Beispiel einer minimalen DSR

```
void dsr(  
    cyg_vector_t vector,  
    cyg_ucount32 count,  
    cyg_addrword_t data)  
{  
    ...  
}
```

- 1 Anzahl der ISRs, die diese DSR anforderten
 ~> normalerweise 1



Beispiel einer minimalen DSR

```
void dsr(  
    cyg_vector_t vector,  
    cyg_ucount32 count,  
    cyg_addrword_t data)  
{  
    ...  
}
```

- 1 Anzahl der ISRs, die diese DSR anforderten
~> normalerweise 1
- 2 Ausführung *synchron* zum Scheduler
Was bedeutet das?



- 1 Einführung in eCos
- 2 eCos-Threads
 - Zeit in eCos
- 3 Interruptbehandlung
 - Einschub: Schlüsselwort volatile
 - ISR & DSR
 - eCos-Unterbrechungsbehandlung
- 4 Periodische Ausführung – eCos Alarme**
- 5 Ereignisse in eCos
 - Events
 - Mailbox
- 6 Zugriffskontrolle in eCos
- 7 Entwicklungsumgebung
- 8 Debuggen mit GDB



eCos-*Alarmer* basieren auf eCos-Zählern (Counter³)

- Anlegen eines Zähler für bestimmtes Ereignis

```
void cyg_counter_create(cyg_handle_t* handle,  
                       cyg_counter* counter)
```

- Inkrementieren:

```
void cyg_counter_tick(cyg_handle_t counter)
```

- Zeitgeberunterbrechung (→ DSR)
 - eCos-interne Uhr als Zähler
- externes Ereignis (Taster, etc.)
 - Zähler wird „von Hand“ inkrementiert (→ DSR, → Faden)

- eCos verwaltet Zählerstand **intern**

- Zugriff auf Zählerstand:

```
cyg_tick_count_t cyg_counter_current_value(cyg_handle_t ctr);  
void cyg_counter_set_value(cyg_handle_t ctr, cyg_tick_count_t val);
```

³<http://ecos.sourceware.org/docs-la/ref/kernel-counters.html>

eCos-Uhren (Clocks⁴) sind spezialisierte Zähler

- Basierend auf *Zeitgeberunterbrechung*
- Festgelegte Zeitauflösung beim Erstellen
- Einzige vorgegebene Uhr: die eCos Uhr

```
cyg_handle_t cyg_real_time_clock(void);
```

- Abfragen:

```
cyg_tick_count_t cyg_current_time(void)
```

- Handle auf Uhr-internen Zähler holen

```
void cyg_clock_to_counter(cyg_handle_t clock,  
                          cyg_handle_t* counter);
```

- Inkrementieren:

```
void cyg_counter_tick(cyg_handle_t counter);
```

- Uhr anlegen: `cyg_clock_create()`:

→ Anwendung ist fürs Inkrementieren zuständig

⁴<http://ecos.sourceware.org/docs-latest/ref/kernel-clocks.html>

eCos-Alarm⁵ führt Aktion bei Erreichen eines Zählerstandes aus

1 Anlegen:

```
void cyg_alarm_create(cyg_handle_t counter,  
                    cyg_alarm_t* alarmfn,  
                    cyg_addrword_t data,  
                    cyg_handle_t* handle,  
                    cyg_alarm* alarm);
```

- counter zugeordneter Zähler
- alarmfn Alarmbehandlung (Funktionspointer)
- data Parameter für Alarmbehandlung
- handle Alarm Handle (vgl. Threaderzeugung)
- alarm Speicher für Alarmobjekt (vgl. Threaderzeugung)


⁵<http://ecos.sourceware.org/docs-latest/ref/kernel-alarms.html>

eCos-Alarm⁵ führt Aktion bei Erreichen eines Zählerstandes aus

1 Anlegen:

```
void cyg_alarm_create(cyg_handle_t counter,  
                    cyg_alarm_t* alarmfn,  
                    cyg_addrword_t data,  
                    cyg_handle_t* handle,  
                    cyg_alarm* alarm);
```

- counter zugeordneter Zähler
- alarmfn Alarmbehandlung (Funktionspointer)
- data Parameter für Alarmbehandlung
- handle Alarm Handle (vgl. Threaderzeugung)
- alarm Speicher für Alarmobjekt (vgl. Threaderzeugung)

 alarmfn wird im DSR-Kontext ausgeführt
→ cyg_thread_resume()

⁵<http://ecos.sourceware.org/docs-latest/ref/kernel-alarms.html>

eCos-Alarm⁶ führt Aktion bei Erreichen eines Zählerstandes aus

2 Alarminitialisierung:

```
void cyg_alarm_initialize(cyg_handle_t alarm,  
                        cyg_tick_count_t trigger,  
                        cyg_tick_count_t interval);
```

- alarm Alarmhandle
- trigger *Absolute* Zählerticks der *ersten* Aktivierung
 - Nutze `cyg_current_time() + x` \rightsquigarrow *Phase*
 - **Vorsicht:** `cyg_current_time()` kann bei jedem Aufruf anderen Wert liefern
 - trigger **muss** in der Zukunft liegen. **Warum?**
- interval Zählerintervall für folgende *periodische* Aktivierungen

3 Alarm freischalten

```
void cyg_alarm_enable(cyg_handle_t alarm)
```

⁶<http://ecos.sourceware.org/docs-latest/ref/kernel-alarms.html>

- 1 Einführung in eCos
- 2 eCos-Threads
 - Zeit in eCos
- 3 Interruptbehandlung
 - Einschub: Schlüsselwort volatile
 - ISR & DSR
 - eCos-Unterbrechungsbehandlung
- 4 Periodische Ausführung – eCos Alarme
- 5 Ereignisse in eCos**
 - Events
 - Mailbox
- 6 Zugriffskontrolle in eCos
- 7 Entwicklungsumgebung
- 8 Debuggen mit GDB



Signalisieren von Ereignissen

- Signale unterstützen *Produzent-Konsument Muster*
- Thread/DSR *signalisiert* Ereignis (z. B. Tastendruck)
... konsumierender Thread *wartet*
- Umsetzung: 32-bit Integer \rightsquigarrow 32 *Einzel-signale* pro Flag
 - Ein Flag erlaubt somit $2^{32} - 1$ Signalkombinationen
 - Threads können auf ein Signalmuster blockierend warten oder pollen

Achtung:

Flags zählen keine Ereignisse! (vgl. HW-Interrupts)

⁷<http://ecos.sourceware.org/docs-latest/ref/kernel-flags.html>

- Produzenten/Konsumenten teilen sich eine Flag-Objekt
- Dieses wird von der *Anwendung* bereitgestellt (vgl. Alarmobjekt)
- Flag-Objekt muss initialisiert werden:

```
cyg_flag_init(cyg_flag_t* flag)
```

- Signal(e) im Flag setzen:

```
cyg_flag_setbits(cyg_flag_t* flag, cyg_flag_value_t value)
```

- Bzw. zurücksetzen:

```
cyg_flag_maskbits(cyg_flag_t* flag, cyg_flag_value_t value)
```

- Auf Signal warten/pollen:

```
cyg_flag_value_t cyg_flag_wait/poll(cyg_flag_t* flag,  
                                     cyg_flag_value_t pattern,  
                                     cyg_flag_mode_t mode);
```



- `cyg_flag_value_t` pattern setzt gewünschte Signalkombination
- `cyg_flag_mode_t` legt Weckmuster fest
 - `CYG_FLAG_WAITMODE_AND`: Alle konfigurierten Signale müssen aktiv sein; Sie bleiben nach dem Aufwachen gesetzt.
 - `CYG_FLAG_WAITMODE_OR`: Mindestens eines der konfigurierten Signale muss aktiv sein; Alle Signale bleiben nach dem Aufwachen gesetzt.
 - `CYG_FLAG_WAITMODE_OR` | `CYG_FLAG_WAITMODE_CLR`: Mindestens eines der konfigurierten Signale muss aktiv sein; Alle gesetzten Signale werden nach dem Aufwachen gelöscht.



```
static cyg_flag_t flag0;

void my_dsr(cyg_vector_t v,
           cyg_ucount32 c,
           cyg_addrword_t d){
    cyg_flag_setbits(&flag0, 0x02);
}

void user_thread(cyg_addr_t data){
    while(true) {
        cyg_flag_wait(&flag0, 0x22,
                    CYG_FLAG_WAITMODE_OR | CYG_FLAG_WAITMODE_CLR);
        printf("Event!\n");
    }
}

void cyg_user_start(void){
    ...
    cyg_flag_init(&flag0);
    ...
}
```





- Zwischen Threads können *Nachrichten* versendet werden
- Konsument erzeugt einen Briefkasten (mailbox) fester Größe
- Produzenten legt Nachrichten dort ab
 - Inhalt: Zeiger auf beliebige Datenstruktur
 - Konsument kann auf *Nachrichtenenpfang* blockieren
 - Produzent blockiert, falls Briefkasten *voll*
 - Aber auch *nicht-blockierende* Aufrufvarianten

⁸<http://ecos.sourceforge.org/docs-latest/ref/kernel-mail-boxes.html>

- Mailbox anlegen:

```
cyg_mbox_create(cyg_handle_t* handle, cyg_mbox* mbox);
```

- Nachricht verschicken:

```
cyg_bool_t cyg_mbox_put(cyg_handle_t mbox, void* item);
```

- Nachricht empfangen:

```
void* cyg_mbox_get(cyg_handle_t mbox);
```

- Empfang *und* Versand können blockieren.

- *try*-Versionen: Würde ich blockieren?

- *timed*-Versionen: Blockieren, aber nur für bestimmte Zeit.

→ Selbststudium!

⁹<http://ecos.sourceware.org/docs-latest/ref/kernel-mail-boxes.html>



Versenden von Nachrichten – Beispiel

■ Initialisierung:

```
static cyg_handle_t mailbox_handle;
static cyg_mbox      mailbox;
void cyg_user_start(void) {
    cyg_mbox_create(&mailbox_handle, &mailbox);
    ...
}
```

■ Produzent (Sender):

```
void producer_entry(cyg_addrword_t data) {
    ...
    cyg_mbox_put(mailbox_handle, &my_message);
    ...
}
```

■ Konsument (Empfänger):

```
void consumer_entry(cyg_addrword_t data) {
    ...
    void *message = cyg_mbox_get(mailbox_handle);
    ...
}
```



- 1 Einführung in eCos
- 2 eCos-Threads
 - Zeit in eCos
- 3 Interruptbehandlung
 - Einschub: Schlüsselwort volatile
 - ISR & DSR
 - eCos-Unterbrechungsbehandlung
- 4 Periodische Ausführung – eCos Alarme
- 5 Ereignisse in eCos
 - Events
 - Mailbox
- 6 Zugriffskontrolle in eCos**
- 7 Entwicklungsumgebung
- 8 Debuggen mit GDB



Kerneldatenstrukturen durch Sperren des Schedulers geschützt

~> **Big Kernel Lock** (BKL)

- Sperre: `void cyg_scheduler_lock(void);`
 - Sofortiges Anhalten des Scheduling
 - Verzögerung der DSR-Ausführungen
 - **ISRs werden weiterhin zugestellt!**
- Freigabe: `void cyg_scheduler_unlock(void);`
 - Sofortige Abarbeitung angelaufener DSRs
- Alle **Systemaufrufe** werden per NPCS synchronisiert
- Anwendungen sollten Mutexe, Semaphore, etc. nutzen
 - **Ausnahme:** Synchronisation zwischen DSR und Thread

¹⁰<http://ecos.sourceware.org/docs-latest/ref/kernel-schedcontrol.html>

Gegenseitiger Ausschluss – eCos-NPCS¹⁰

Nicht-preemptiver kritischer Abschnitt durch Sperren des Schedulers

Kerneldatenstrukturen durch Sperren des Schedulers geschützt

~> **Big Kernel Lock** (BKL)

- Sperre: `void cyg_scheduler_lock(void);`
 - Sofortiges Anhalten des Scheduling
 - Verzögerung der DSR-Ausführungen
 - **ISRs werden weiterhin zugestellt!**
- Freigabe: `void cyg_scheduler_unlock(void);`
 - Sofortige Abarbeitung angelaufener DSRs
- Alle **Systemaufrufe** werden per NPCS synchronisiert
- Anwendungen sollten Mutexe, Semaphore, etc. nutzen
 - **Ausnahme:** Synchronisation zwischen DSR und Thread

Was sind die Vor- bzw. Nachteile des BKL Konzepts?

¹⁰<http://ecos.sourceware.org/docs-latest/ref/kernel-schedcontrol.html>



■ Initialisierung

```
void cyg_mutex_init(cyg_mutex_t* mutex);
```

■ Protokoll auswählen:

```
void cyg_mutex_set_protocol(cyg_mutex_t* mutex,  
                           enum cyg_mutex_protocol protocol);
```

- CYG_MUTEX_NONE keine Prioritätsvererbung
- CYG_MUTEX_INHERIT erbe Priorität des aktuellen Inhabers
- CYG_MUTEX_CEILING erbe Prioritätsobergrenze

■ nur bei CYG_MUTEX_CEILING: Prioritätsobergrenze setzen

```
void cyg_mutex_set_ceiling(cyg_mutex_t* mutex,  
                           cyg_priority_t priority);
```

■ *Prioritätsobergrenze +1 höherprior als Thread*

¹¹<http://ecos.sourceware.org/docs-latest/ref/kernel-mutexes.html>

■ Mutex belegen

```
cyg_bool_t cyg_mutex_lock(cyg_mutex_t* mutex);
```

Rückgabewert

- true falls Belegen erfolgreich
- false sonst

■ Mutex freigeben:

```
void cyg_mutex_unlock(cyg_mutex_t* mutex);
```



```
static cyg_mutex_t s_mutex;

void cyg_user_start(void) {
    // Mutex initialisieren
    cyg_mutex_init(&s_mutex);

    // Protokoll auswaehlen
    cyg_mutex_set_protocol(&s_mutex, CYG_MUTEX_CEILING);

    // Prioritaetsobergrenze festlegen
    cyg_mutex_set_ceiling(&s_mutex, 3);

    // Tasks, Alarme etc.
}

void task_entry(cyg_addrword_t data) {
    cyg_mutex_lock(&s_mutex); // auf Freigabe warten
    // kritischer Abschnitt
    cyg_mutex_unlock(&s_mutex); // Mutex freigeben
}
```



- 1 Einführung in eCos
- 2 eCos-Threads
 - Zeit in eCos
- 3 Interruptbehandlung
 - Einschub: Schlüsselwort volatile
 - ISR & DSR
 - eCos-Unterbrechungsbehandlung
- 4 Periodische Ausführung – eCos Alarme
- 5 Ereignisse in eCos
 - Events
 - Mailbox
- 6 Zugriffskontrolle in eCos
- 7 Entwicklungsumgebung**
- 8 Debuggen mit GDB



EZS-Wiki

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

- Umstellung auf neues Entwicklungsboard

~> Mögliche Probleme



Dokumentation im Wiki

- Erweiterung durch *alle Teilnehmer*

- gitlab account notwendig

- Besonders wertvolle Beiträge

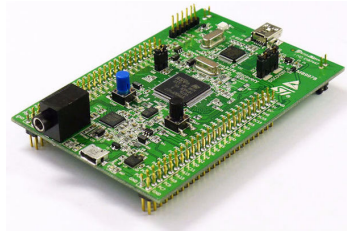


Gutscheine für I4-Kaffeekarte

- VM für weitere Plattformen (Windows, OSX...)



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



- 1 Vorgabe herunterladen, entpacken, Verzeichnis betreten
- 2 **Nötige Umgebungsvariablen setzen:** `source ecosenv.sh`
- 3 Eigene Quelldateien (**nur**) in `CMakeLists.txt` eintragen
- 4 `build` Verzeichnis betreten → out-of-source build¹²
- 5 Makefiles erzeugen: `cmake ..` (*cmake PUNKT PUNKT*)
- 6 Alles kompilieren: `make`
- 7 Flashen, ausführen, debuggen: `make flash`
~> Flasher & Debugger: `make gdb` (später ausführlicher)

¹²www.cmake.org/Wiki/CMake_FAQ#Out-of-source_build_trees



- 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 Firmware¹³ bereits ausgestattet

¹³<https://github.com/blacksphere/blackmagic>

- Bashprompt: , gdb-Prompt: >

```
cd <ezs-aufgabe1>
```

```
source ecosenv.sh
```

```
cd build
```

```
cmake ..
```

```
make ##Erstellen der Software
```

```
make flash ##Aufspielen der Software
```

```
oder
```

```
make debug ##Aufspielen und Debuggen mittels gdb
```



- 1 Einführung in eCos
- 2 eCos-Threads
 - Zeit in eCos
- 3 Interruptbehandlung
 - Einschub: Schlüsselwort volatile
 - ISR & DSR
 - eCos-Unterbrechungsbehandlung
- 4 Periodische Ausführung – eCos Alarme
- 5 Ereignisse in eCos
 - Events
 - Mailbox
- 6 Zugriffskontrolle in eCos
- 7 Entwicklungsumgebung
- 8 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
38 res_ps = (PRESCALER+1) * 1000000L / RCCLOCK;
39 res_us = res_ps / 1000000L;
40 }
41 cyg_uint64 ezs_counter_get(void) {
42     return timer_get_counter(TIM5);
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     r0, [pc, #0] ; (0x080045c <ezs_counter_get()+12>)
0x08000454 ezs_counter_get+4 bl     0x0800f00 <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/libEZ5/drivers/stm32f4/ezs_counter.cpp:41
(no arguments)
[1] from 0x080004e8 in ezs_delay_us+110 at /home/noctux/work/ezs-aufgaben/ezs-aufgaben/HelloWorld/libEZ5/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/libEZ5/drivers/stm32f4/ezs_counter.cpp:41
--- Locals
ezs_counter_get () at /home/noctux/work/ezs-aufgaben/ezs-aufgaben/HelloWorld/libEZ5/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 0x12a9 lma 0x8000938
Loading section .data, size 0x4a0 lma 0x8000bbe8
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(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(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. Attc Driver
  3  STM32F4xx
-- Source
36 res_ps = (PRESCALER+1) * 100000L / RCCLOCK;
37 res_us = res_ps / 100000L;
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]          ; (0x080045c <ezs_counter_get()+12>)
0x08000454 ezs_counter_get+4  bl      0x0800f00 <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 0x0800046e 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 0x12e8 lma 0x800d938
Loading section .data, size 0x4e8 lma 0x800e8e8
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) % (100000/delay_us) == 0)
41         printf("Hallo Welt!\n");
42
43     up = not up;
44     ezs_gpio_set(up);
45
46 }
Assembly
0x080000e4 test_thread+20 ldr.w r8, [pc, #132] ; 0x080016c <test_thread+156>
0x080000e8 test_thread+24 ldr r7, [pc, #108] ; (0x0800158 <test_thread+136>)
0x080000ea test_thread+26 ldr r6, [pc, #112] ; (0x080015c <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-aufgaben/scripts/gen_14ezs/files/ecos/packages/kernel/cURRENT/src/common/thread_cxx:94
arg thread = 0x20000730 <threaddata>
(-)
Threads
[1] id 0 from 0x080000ec in test_thread+28 at /home/noctux/work/ezs-aufgaben/ezs-aufgaben/HelloWorld/hello.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
2 breakpoint keep y 0x08000170 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
- Serielle Terminals
 - cutecom (graphisch)
 - minicom (Terminal)
- `minicom -D /dev/ttyACM1` (Strg-A Z für Hilfe/weitere Befehle wie Exit)
- In Dashboard auch `make` & `make flash` verfügbar,
anschließend Programm mit `c(ontinue)` weiter ausführen

