

DIY – Individual Prototyping and Systems Engineering

Embedded Entwicklung

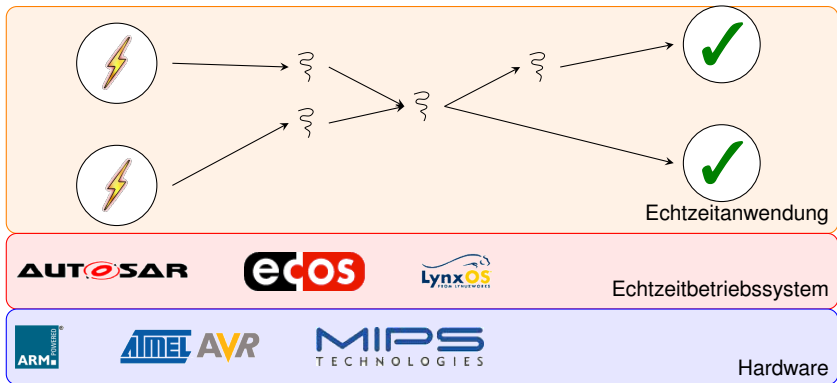
Tobias Klaus

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

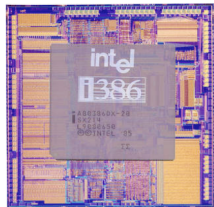
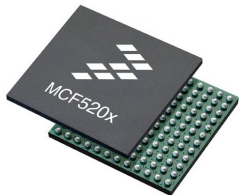
<https://www4.cs.fau.de>

1. Juni 2017



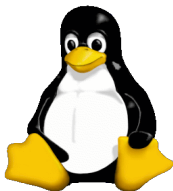


Prozessorvielfalt in der Echtzeitwelt



free **RTOS**

μ Clinux



eCos

QNX



μ C/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>

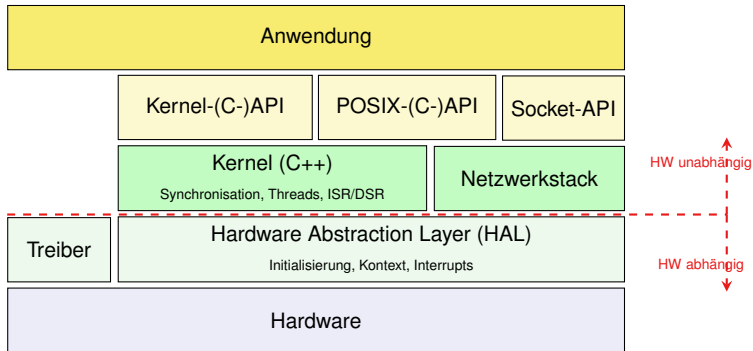


Unterstützte Plattformen

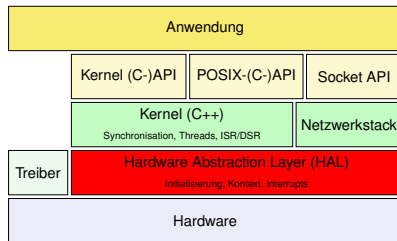
<http://www.ecoscentric.com/ecos/examples.shtml>

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

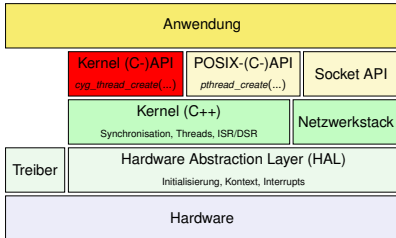




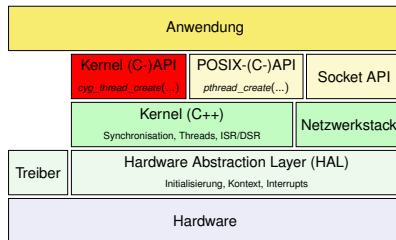
- Abstrahiert CPU- und plattformspezifische Eigenschaften
 - Kontextwechsel
 - Interruptverwaltung
 - CPU-Erkennung, Startup
 - Zeitgeber, I/O-Registerzugriffe



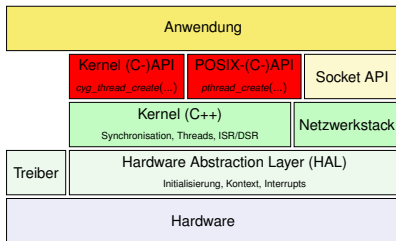
- Implementiert in C++
- Feingranular konfigurierbar
 - Verschiedene Schedulingstrategien (Bitmap/Multilevel Queue)
 - Zeitscheibenbasiert, präemptiv, prioritätenbasiert
- Verschiedene Synchronisationsstrategien
 - Mutexe, Semaphore, Bedingungsvariablen
 - Messages Boxes



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

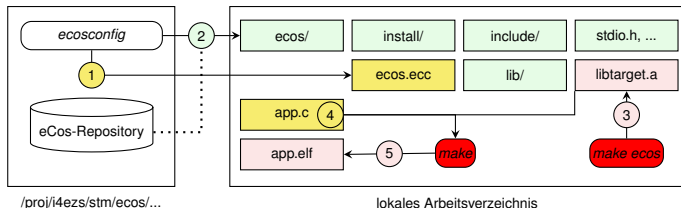


- Kernel API
 - vollständige C-Schnittstelle
 - siehe Dokumentation¹
- (Optionale) POSIX-Kompatibilitätsschicht
 - Scheduling-Konfiguration, *pthread*_*
 - Timer, Semaphore, Message Queues, Signale, ...



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

- 1 Erstellen einer Konfiguration (configtool/ecosconfig)
- 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

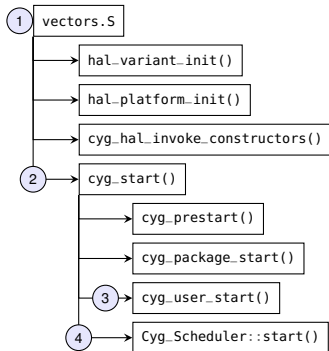
2 cyg_start():

- Hardwareunabhängige Vorbereitungen

3 cyg_user_start():

- Einsprungpunkt für Anwendungscode!
- Erzeugen von Threads

4 Starten des Schedulers



Wichtig!

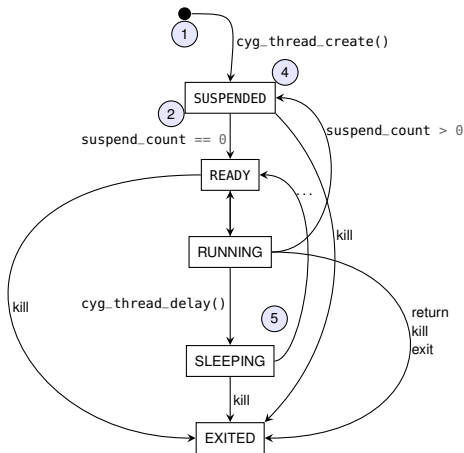
`cyg_user_start()` muss zurückkehren!



- 1 Einführung in eCos
- 2 eCos-Threads**
- 3 Interruptbehandlung
 - ISR & DSR
 - eCos-Unterbrechungsbehandlung
- 4 Periodische Ausführung – eCos Alarme
- 5 Ereignisse in eCos
 - Events
 - Mailbox
- 6 Zugriffskontrolle in eCos
- 7 Entwicklungsumgebung
- 8 Hardwareprogrammierung



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

Ausführliche Dokumentation

<http://ecos.sourceforge.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
);
```

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

Ausführliche Dokumentation

<http://ecos.sourceforge.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
);
```

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

Ausführliche Dokumentation

<http://ecos.sourceforge.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
);
```

- 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_ucount32 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_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?*

Ausführliche Dokumentation

<http://ecos.sourceforge.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>



```
#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);
}
```



- 1 Einführung in eCos
- 2 eCos-Threads
- 3 Interruptbehandlung**
 - ISR & DSR
 - eCos-Unterbrechungsbehandlung
- 4 Periodische Ausführung – eCos Alarme
- 5 Ereignisse in eCos
 - Events
 - Mailbox
- 6 Zugriffskontrolle in eCos
- 7 Entwicklungsumgebung
- 8 Hardwareprogrammierung



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_acknowledge_isr(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 ISR

```
cyg_uint32 isr(cyg_vector_t vector, cyg_addrword_t data) {
    cyg_bool_t dsr_required = 0;
    ...
    cyg_acknowledge_isr(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 ISR

```
cyg_uint32 isr(cyg_vector_t vector, cyg_addrword_t data) {
    cyg_bool_t dsr_required = 0;
    ...
    cyg_acknowledge_isr(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 ISR

```
cyg_uint32 isr(cyg_vector_t vector, cyg_addrword_t data) {
    cyg_bool_t dsr_required = 0;
    ...
    cyg_acknowledge_isr(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_function(  
    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?



Beispiel einer minimalen DSR

```
void dsr_function(  
  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
- 3 Interruptbehandlung
 - ISR & DSR
 - eCos-Unterbrechungsbehandlung
- 4 Periodische Ausführung – eCos Alarme**
- 5 Ereignisse in eCos
 - Events
 - Mailbox
- 6 Zugriffskontrolle in eCos
- 7 Entwicklungsumgebung
- 8 Hardwareprogrammierung



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)

 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
- 3 Interruptbehandlung
 - ISR & DSR
 - eCos-Unterbrechungsbehandlung
- 4 Periodische Ausführung – eCos Alarme
- 5 Ereignisse in eCos**
 - Events
 - Mailbox
- 6 Zugriffskontrolle in eCos
- 7 Entwicklungsumgebung
- 8 Hardwareprogrammierung



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.sourceware.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
- 3 Interruptbehandlung
 - ISR & DSR
 - eCos-Unterbrechungsbehandlung
- 4 Periodische Ausführung – eCos Alarme
- 5 Ereignisse in eCos
 - Events
 - Mailbox
- 6 Zugriffskontrolle in eCos**
- 7 Entwicklungsumgebung
- 8 Hardwareprogrammierung



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>

■ 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
- 3 Interruptbehandlung
 - ISR & DSR
 - eCos-Unterbrechungsbehandlung
- 4 Periodische Ausführung – eCos Alarme
- 5 Ereignisse in eCos
 - Events
 - Mailbox
- 6 Zugriffskontrolle in eCos
- 7 Entwicklungsumgebung**
- 8 Hardwareprogrammierung



EZS-Wiki

<https://gitlab.cs.fau.de/ezs/ezs16/wikis/home>

- Umstellung auf neues Entwicklungsboard

~> Mögliche Probleme



Dokumentation im Wiki

- Erweiterung durch *alle Teilnehmer*

- gitlab account notwendig

- Besonders wertvolle Beiträge

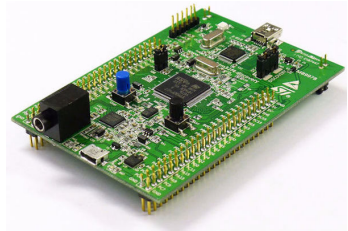


Gutscheine für I4-Kaffeekarte

- 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
```

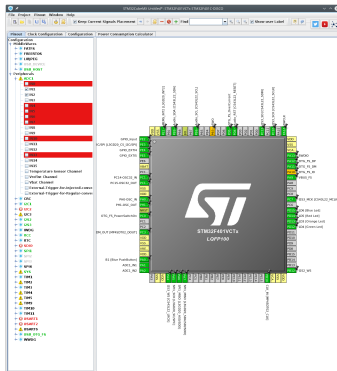
oder

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



- 1 Einführung in eCos
- 2 eCos-Threads
- 3 Interruptbehandlung
 - ISR & DSR
 - eCos-Unterbrechungsbehandlung
- 4 Periodische Ausführung – eCos Alarme
- 5 Ereignisse in eCos
 - Events
 - Mailbox
- 6 Zugriffskontrolle in eCos
- 7 Entwicklungsumgebung
- 8 Hardwareprogrammierung**





→ http://www.st.com/content/st_com/en/products/development-tools/software-development-tools/stm32-software-development-tools/stm32-configurators-and-code-generators/stm32cubemx.html



```
static void MX_ADC1_Init(void)
{
    ADC_HandleTypeDef hadc1;

    ADC_ChannelConfTypeDef sConfig;
    hadc1.Instance = ADC1;
    hadc1.Init.ClockPrescaler = ADC_CLOCK_SYNC_PCLK_DIV4;
    hadc1.Init.Resolution = ADC_RESOLUTION_12B;
    ...
    if (HAL_ADC_Init(&hadc1) != HAL_OK) {
        error();
    }
    sConfig.Channel = ADC_CHANNEL_1;
    sConfig.Rank = 1;
    ...
    if (HAL_ADC_ConfigChannel(&hadc1, &sConfig) != HAL_OK) {
        error();
    }
}
```



■ Drivers/STM32F4xx_HAL_Driver/Inc/stm32f4xx_hal_adc.h

```
typedef struct {
    ADC_TypeDef      *Instance;           /* Register base address */
    ADC_InitTypeDef  Init;               /* ADC required parameters */
    __IO uint32_t    NbrOfCurrentConversionRank; /* ADC number of current conversions */
    DMA_HandleTypeDef *DMA_Handle;       /* Pointer DMA Handler */
    HAL_LockTypeDef  Lock;               /* ADC locking object */
    __IO uint32_t    State;               /* ADC communication state */
    __IO uint32_t    ErrorCode;          /* ADC Error code */
}ADC_HandleTypeDef;
```

■ Drivers/CMSIS/Device/ST/STM32F4xx/Include/stm32f401xc.h:

```
/*!< Peripheral memory map */
#define APB1PERIPH_BASE    PERIPH_BASE
#define APB2PERIPH_BASE    (PERIPH_BASE + 0x00010000U)
...
#define ADC1_BASE          (APB2PERIPH_BASE + 0x2000U)
#define ADC1_COMMON_BASE   (APB2PERIPH_BASE + 0x2300U)
...
#define ADC1                ((ADC_TypeDef *) ADC1_BASE)
#define ADC1_COMMON          ((ADC_Common_TypeDef *) ADC1_COMMON_BASE)
```



- Datenblätter:

http://www.st.com/content/st_com/en/products/microcontrollers/stm32-32-bit-arm-cortex-mcus/stm32f4-series/stm32f401/stm32f401cb.html

- Hal-Beschreibung und Beispiele(stm32cubef4):

http://www.st.com/content/st_com/en/products/embedded-software/mcus-embedded-software/stm32-embedded-software/stm32cube-embedded-software/stm32cubef4.html

 Sehr Umfangreich! Hal Beschreibung: 1838 S.!

- STM32CubeMX: Graphische Software zur Generierung von Treibercode:

http://www.st.com/content/st_com/en/products/development-tools/software-development-tools/stm32-software-development-tools/stm32-configurators-and-code-generators/stm32cubemx.html

- libEZS(Beispielanwendung): libEZS/include/drivers/stm32f4/*

