# Advanced Embedded Systems

Zusammenfassung

Joel von Rotz & Andreas Ming 23.05.2023

Quelldateien

## Inhaltsverzeichnis

- ☐ FreeRTOS
  - ☐ Task
  - ☐ Scheduler
  - ☐ Priority
  - ☐ Interrupts
  - ☐ Semaphore & Mutex
  - ☐ Timer
  - ☐ Queue

# Entwicklung

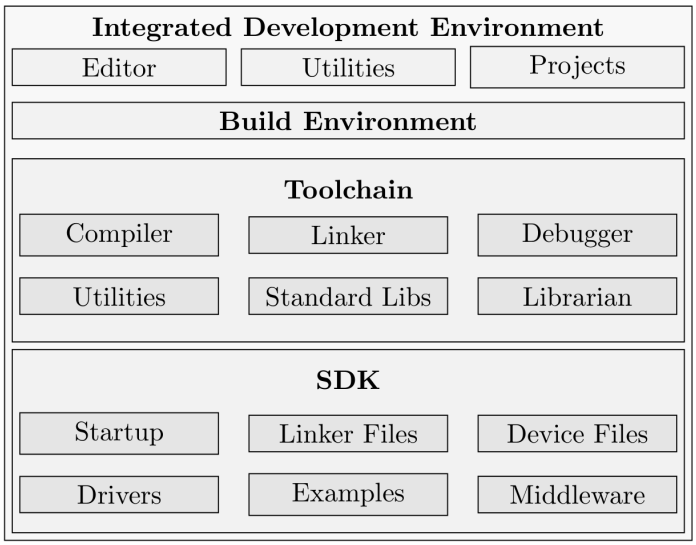
## Cross-Development

*Cross-Development* bedeutet die Entwicklung einer Firmware auf einem **Host** für einen **Target**. Grund dafür ist, dass das *embedded* Target nicht genügend Ressourcen (CPU Leistung, Speicher) für die direkte Entwicklung hat.

> **Target & Host**
>
> **Target** (wofür): Zielsystem, für das man entwickelt.
> **Host** (womit): bezeichnet die Umgebung , auf der man die Entwicklung vornimmt.

### Integrated Development Environment

Eine IDE besteht aus vier Hauptteilen: IDE spezifische Funktionen, die Build Enivronment, die (GNU-)Toolchain und die SDK des entsprechenden Targets.




**Toolchain**: Kollektion von Tools wie Compiler, Linker, Debugger, etc. → einzelne Werkzeuge zum Zusammensetzen der Firmware

**Build Environment**: Steuert die Toolchain und den Übersetzungsvorgang → *make*, *Makefiles*

**IDE**: "Fancy Editor", beinhaltet Tools für bessere Produktivität, wendet Build Environment an → Intellisense, Workspace, Projekte

**SDK**: Software Development Kit → Treiber (UART, $I^2C$, SPI,...), Beispiele (Board spezifisch), Projekt und Debugger Konfiguration (CMSIS-SVD, CMSIS-DAP,...), Device Files (Liste von Register und deren Adressen)

## Eclipse (Open Source IDE)

Plugin-basierter Editor → deckt mehrere Programmiersprachen und Environments ab.

\+ Sehr modular (Plugin System), kann auf eigenen Workflow (ungefähr) angepasst werden ; als IDE vereinfacht die Entwicklung ;

\- Eierlegende Wollmilchsau → kann zu viel als nötig ist (abhängig von Workflow und Funktionsumfang) ;

> **Geschichte**
>
> IDE wurde hauptsächlich von IBM (International Business Machines) auf der Code Basis vom *VisualAge IDE* in 2001 entwickelt und später mit Zusammenarbeit (Konsortium) von *Borland*, *QNX*, *Red Hat*, *SuSe* und andere entstand Eclipse.
> → Grund für Erfolg war das Plugin System und die Anpassbarkeit

### Plugin System

Haupt-Gimmick von Eclipse ist das Plugin System, welches die Erweiterung der bestehenden Entwicklungsumgebung durch weitere *Werkzeuge* wie zum Beispiel *Hex Editor* erlaubt.
→ Ermöglicht eine feinere Anpassung der Entwicklungsumgebung

### Workspace

Eclipse IDE arbeitet mit *Workspaces* → Kollektion von Projekten und Einstellungen (aktive Plugins, verwendete Version, spezifische Kompiler Einstellungen).

> **Warnung**
>
> Pro IDE Version ein eigener Workspace → wegen Versionskonflikte

### Begriffe

**Workspace** – Arbeitsplatz, Kollektion von Projekten, Einstellungen und aktive Plugins

**Views** – Einzelne Module/Fenster (z.B. *Variables* oder *FreeRTOS Task View*)

**Perspectives** – vordefinierte Gruppe & Platzierung von Views (z.B. Debug, Develop,. . . )

# Systeme

**!** Was ist ein *embedded* System?

Ein Rechner (CPU, MCU, etc.) integriert in ein System. Für eine Aufgabe/Zweck optimiert und meistens **kein** normaler Computer! Meistens von aussen nicht direkt zugreifbar, anders als beim Computer.
Anwendung: Echtzeitsystem, Wetterstation, Steuerung für Roboterarm, etc.

## Transformierende Systeme

Verarbeitet ein Eingabesignal (*Input*) und gibt ein Ausgabesignal (*Output*) aus. Wichtige Charakteristiken sind **Verarbeitungsqualität** (effiziente Datenverarbeitung), **Durchsatz** (kleine Latenz zwischen In- und Output), **optimierte Systemlast** (ein System, welches für die Aufgabe ausgelegt ist ; nicht überdimensioniert) und **optimierter Speicherverbrauch** (wenig Speicher bedeutet meistens auch langsames System, daher muss Speicher effizient gebraucht werden).

Beispiele: Verschlüsselung, Router, Noise Canceling, MP3/MPEG En-/Decoder

## Reaktive Systeme

Ein Reaktives System reagiert auf gemessene Werte, also von externen Events. Das System muss eine **kurze Reaktionszeit** garantieren, da meistens solche Systeme für Notfallsituationen verwendet werden. Ebenfalls werden diese für **Regelkreise** verwendet und sind typisch **Echtzeitsysteme**.

Beispiele: Airbag, Roll-Over Detection, ABS, Brake Assistance, Engine Control, Motorsteuerung

## Interaktive Systeme

Interaktive Systeme werden von Benutzer interagiert. Sie haben eine **hohe Systemlast**, da zum Beispiel die Interaktion einer Benutzeroberfläche ausgewertet werden muss. Damit ein Benutzer mit dem System interagiert, muss es ein **optimiertes HMI** (Human-Machine-Interface) sein und eine **'kurze' Antwortzeit**.

Beispiele: Ticket-Automat, Taschenrechner, Smart-Phone, Fernsehbedienung

## Kombiniertes System

Ein kombiniertes System ist, wär hätte es gedacht, eine Kombination von den erwähnten Systemen und anderen. Zum Beispiel kann ein Smartphone ein kombiniertes System ist, da es aus einem interaktiven Teilsystem für Homescreen- & App-Interaktionen, transformierendes Teilsystem für Audio-Decodierung für Musikhören und weiteren kleineren Teilsystemen.

# Mikrocontroller

## ARM Cortex Familie

# FreeRTOS

## Echtzeit

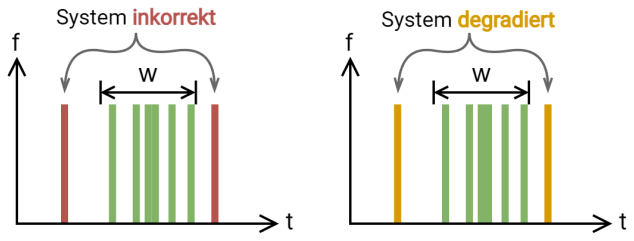
**!** Was ist Echtzeit?

Ein Computer ist als Echtzeitsystem klassifiziert, wenn er auf externe Ereignisse in der echten Welt reagieren kann: mit dem richtigen Resultat, zur richtigen Zeit, unabhängig der Systemlast, auf eine deterministische und vorhersehbare Weise.

**!**

- **Absolute** Rechtzeitigkeit – Absoluter Zeitpunkt (z.B. jeden Tag 05:30 ± 1 Minute)
- **Relative** Rechtzeitigkeit – Relative Zeit nach Ereignis (z.B. 5 Minuten ± nach Einschalten wieder ausschalten)

### Harte & Weiche Echtzeit




- **Harte** Echtzeit (links) – Zeitbedingung einhalten (innerhalb Zeitfenster $w$). **Beispiel** Airbag soll 20ms nach Aufpralldetektion ausgelöst werden.

- **Weiche** Echtzeit (rechts) – Immer noch in Ordnung, wenn Zeitbedingung nicht eingehalten. **Beispiel** Video Encoder wiedergibt mit Framerate 25 F/s. Framerate darf nicht unter 10 F/s sein und in 10% der Zeit Framerate unter 25 F/s → System ist immer noch als korrekt angesehen.

### FreeRTOS

### Interrupts

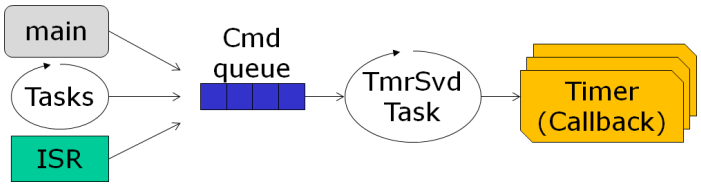
### Kernel

### Task

#### IDLE-Task

### Timer

#### afd




> Timer Service Daemon
>
> Mit `configUSE_TIMERS` wird die Timer-Funktionen aktiviert und aktiviert automatisch die *Timer Service Daemon*

### Queue

### Semaphore & Mutex

### Prioritäten

## C - Konzepte

### Synchronisation

#### Realtime

#### Gadfly / Polling

*Gadfly* Sync überprüft periodisch einen Zustand und fährt erst dann weiter, wenn die Bedingung erfüllt ist.

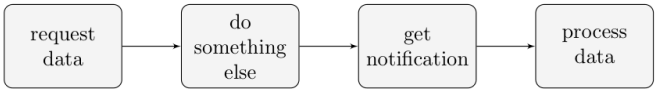
```c
void read(void) {
  for (size_t i = 0; i < sizeof(buffer); i++) {
    while (!PORTB.B0) { /* reading 0: no hole */
      /* while there is no hole , wait for
      ↪  rising edge */
    }
    buffer[i] = PORTA;

    /* in the hole: read data */
    while (PORTB.B0) {
      /* reading 1: we have a hole */
      /* get out of hole , wait for falling edge
      ↪  */
    }
  }
}
```

\+ Benötigt keine unnötige Rechenzeit ;

\- Benötigt keine unnötige Rechenzeit

#### Interrupt




**Signalisierung → Zustand sichern → Verzweigung → Rettung benutzter Register → ISR Programm → Exit ISR → Rückkehr zum unterbrochenen Programm**

## C - Bibliotheken