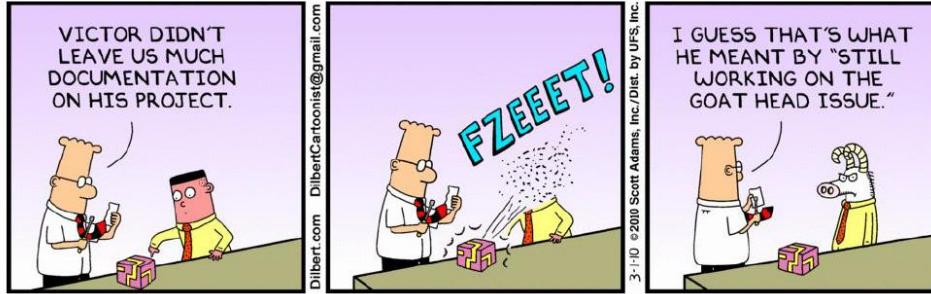


Zusammenfassung INTRO



Merkt euch die Freitagnachmittag-Witze von Herrn Styger ☺

Inhalt

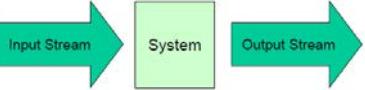
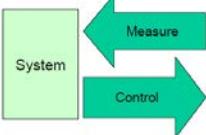
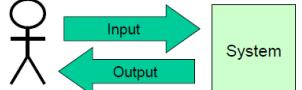
1.	Systems & Realtime	4
1.1.	Typen von Systemen.....	4
1.2.	Realtime Systeme	4
1.2.1.	Timeliness (Rechtzeitigkeit)	4
2.	Macros	5
3.	Header und Source	6
3.1.	Header / Source: Was ist wo?.....	6
3.2.	Funktionen des Header-Files	6
3.3.	Header: Beispiele und Hinweise	6
4.	Synchronization	7
4.1.	Was ist Synchronisation? Wieso benötigt man Synchronisation?	7
4.2.	Arten von Synchronisation	7
4.2.1.	Vergleich der Synchronisationsmethoden	8
5.	Interrupts.....	9
5.1.	Design Hinweise für Interrupts.....	10
5.2.	Reentrancy.....	10
5.2.1.	Sharing Code & Sharing Data.....	10
6.	Critical Sections	11
7.	Doxxygen.....	12
8.	State Machine (e.g. Mealy)	14
9.	Shell	15

9.1. Implementation einer Shell	15
9.2. Shell Protected Access	17
9.2.1. Zugriff auf gemeinsame Ressourcen	17
10. Simple Events	18
11. Clock and Timer	19
11.1. Clock	19
11.2. Timer	19
12. Simple Trigger	20
13. Debounce	22
14. RTOS / FreeRTOS	23
14.1. RTOS	23
14.2. FreeRTOS	25
14.2.1. Ticks	26
14.2.2. Speicher allozieren (für Tasks, Queues und Semaphoren)	26
14.2.3. Tasks	26
14.2.4. Semaphore & Mutex	28
14.2.5. Queues	30
14.2.6. FreeRTOS Trace	31
15. Motor	32
15.1. Motor Signals	32
15.2. Motor Trace	33
15.3. Quadrature Encoder	34
15.4. Quadrature Decoder	34
15.5. Tacho	36
15.6. Closed Loop Control (PID Regulator)	38
16. Accelerometer	40
17. LCD	41
17.1. Bootloader	41
17.2. I2C	41
17.3. C++ and Slider	42
18. Radio Tranceiver	45
18.1. Remote Control	47
19. Wireless (IEEE 802.15.4)	48
19.1. ZigBee	51

20.	Low Power	54
21.	Hardware stuff.....	57
21.1.	Pull-Up und Pull-Down Widerstände.....	57
21.2.	Entprellen	57
21.3.	Verschiedene Devices / Bausteine	58
22.	Compiler and Linker stuff	60
23.	Laboratory Short Courses.....	61
23.1.	Exploring Embedded C.....	61
23.2.	Serial I/O Interfaces – RS-232-C.....	63
23.3.	Interrupts using C	64
23.4.	Analog Input Sampling.....	66
24.	Sonstiges.....	68
24.1.	Stichwörter Example Exam A.....	68
24.2.	Stichwörter Example Exam B.....	68

1. Systems & Realtime

1.1. Typen von Systemen

Transforming Systems	Reactive Systems	Ineractive Systems
 <p>Typische Eigenschaften:</p> <ul style="list-style-type: none"> • Data processing quality • Throughput • Optimized system load • Optimized Memory Usage 	 <p>Typische Eigenschaften:</p> <ul style="list-style-type: none"> • External events are driving system • Guaranteed response time • Control loop • Realtime 	 <p>Typische Eigenschaften:</p> <ul style="list-style-type: none"> • Short response time • High system load • Human-Machine Interaction (HMI)

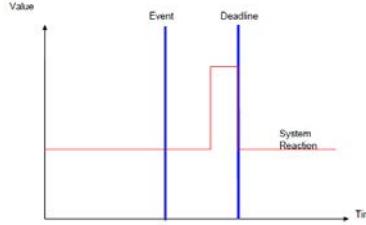
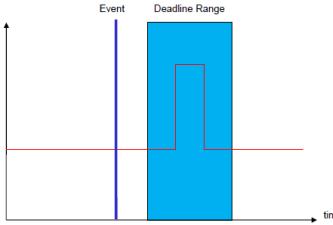
- Vielfach kommt eine Kombination der Systeme zum Einsatz (z.B. Smartphone)

1.2. Realtime Systeme

<ul style="list-style-type: none"> • System interagiert mit der Umgebung • Verschiedene Geschwindigkeitsbereiche / unterschiedliche Häufigkeit der Events • Das System muss mit den Zeitanforderungen der realen Welt umgehen können <p>Anforderungen an ein Realtime System</p> <ul style="list-style-type: none"> • The correct result • At the correct time • Independent of current system load • In a deterministic and foreseeable way 	<p>Realtime heisst nicht „So schnell wie möglich“ sondern</p> <p>„Das richtige Resultat zur richtigen Zeit“!</p>
--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------	--------------------------------------------------------------------------------------------------------------------------------

1.2.1. Timeliness (Rechtzeitigkeit)

Problem: Reale Welt ist nebenläufig, aber Computer arbeiten sequentiell

Reaktionszeit	Hard Realtime	Soft Realtime
<p>Realtime Systeme benötigen eine definierte Reaktionszeit</p> <ul style="list-style-type: none"> • Absolute Zeit • Relative Zeit 	 <p>Inkorrekt falls korrektes Resultat nicht den Zeitanforderungen entspricht</p> <p>→ Ausserhalb des blau markierten Bereiches sind Daten unbrauchbar: Fehler!</p>	 <p>Degradierung falls korrektes Resultat nicht den Zeitanforderungen entspricht</p> <ul style="list-style-type: none"> • Resultat ist ausserhalb des blau markierten Bereichs noch ok aber nicht vollumfänglich „gut“
<p>Interaktive Systeme</p> <ul style="list-style-type: none"> • Sekunden <p>Reaktive & Transitive Systeme</p> <ul style="list-style-type: none"> • Millisekunden • Mikrosekunden <p>System load definiert mit</p> <ul style="list-style-type: none"> • Anzahl nebenläufiger events/tasks • Intervall der Events • Reaktionszeit der Events • Verarbeitungszeit der Events 		

2. Macros

Makros sind grundsätzlich eine textuelle Ersetzung (geschieht durch den Compiler).

Vorteile / Einsatzzwecke	Nachteile
<ul style="list-style-type: none"> • Schnellerer Code • Kleinerer Code • Namen anstatt "magic numbers" (textuelle Ersetzung) <ul style="list-style-type: none"> ◦ <code>#define DELAY_TIME_MS 10</code> • Konfiguration <ul style="list-style-type: none"> ◦ <code>#define DEBUG_ME 1</code> • Portabilität <ul style="list-style-type: none"> ◦ <code>#define ENABLE_INTERRUPT_asm CLI;</code> • Optimierung <ul style="list-style-type: none"> ◦ z.B. auf versch. Plattformen 	<ul style="list-style-type: none"> • Interface • Encapsulation • Debugging • Code kann unleserlich werden • Man muss wirklich wissen was man tut, sonst kann es zu Fehlern während der Laufzeit kommen (siehe Traps & Pitfalls). Der Compiler wird diese Fehler NICHT anzeigen, da reine Programmierfehler!

Traps & Pitfalls / Application hints

```
#define INCI(i) {int a=0; i++;}
void main(void) {
    int a = 0, b = 0;
    INCI(a);
    INCI(b);
    printf("a is now %d, b is now %d\n", a, b);
}
```

a is now 0, b is now 1

```
#define PRE_DELAY 5
#define POST_DELAY 2
#define DELAY      PRE_DELAY + POST_DELAY

return totalDelay(int nofIterations) {
    return nofIterations * DELAY;
}
```

n * 5 + 2

Solution (Klammern verwenden!):

```
#define PRE_DELAY 5
#define POST_DELAY 2
#define DELAY      (PRE_DELAY + POST_DELAY)
```

```
#define PRE_DELAY (5*3)
#define POST_DELAY (2+5)
#define DELAY      ((PRE_DELAY) + (POST_DELAY))

return totalDelay(int nofIterations) {
    return nofIterations * (((5*3)) + ((2+5)));
}
```

Reihenfolge von Befehlen beachten!

Weitere Makro Code-Beispiele	
Plattform-Konfiguration 1	Einfache Outputs setzen
<pre>#define PL_HAS_LED 1 // Kein ; !! #if PL_HAS_LED LED Init(); #endif</pre>	<pre>#if PL_IS_SRB_BOARD #define LED_TURN_ON(nr) (LED##nr##_ClrVal()) #else #define LED_TURN_ON(nr) (LED##nr##_SetVal()) #endif #define LED1_On() (LED_TURN_ON(1)) // obige Zeile wird ersetzt durch LED1_SetVal(); oder LED1_ClrVal(); // → textuelle Ersetzung durch den Compiler #define LED2_On() (LED_TURN_ON(2))</pre>
Plattform-Konfiguration 2	
	<pre>#define PL_HAS_TRIGGER (1 && PL_HAS_TIMER) // Klammern nicht vergessen!</pre>

3. Header und Source

3.1. Header / Source: Was ist wo?

- Deklaration: Name "sichtbar" machen
- Definition: memory allocation
- Konvention:**
 - *.c: Implementation File, Definition
 - *.h: Interface/Header File, Deklaration

```
/* drv.c */
#include "drv.h"
int DRV_global = 7;
static int v;

void DRV_Init(void) {
    v = 3;
    DRV_global += v;
}

/* drv.h */
#ifndef __DRV_H__
#define __DRV_H__
extern int DRV_global;
void DRV_Init(void);
#endif /* __DRV_H__ */

/* main.c */
#include "drv.h"
void main(void) {
    DRV_Init();
}
```

3.2. Funktionen des Header-Files

#include "XX.h" = Textuelle Ersetzung	#ifndef - #define - #endif
	<ul style="list-style-type: none"> Schutz gegen <ul style="list-style-type: none"> Mehrfache Deklarationen / Definitionen Rekursive Includes Namenkonflikte <pre>#ifndef LED_H_ #define LED_H_ #endif</pre>

3.3. Header: Beispiele und Hinweise

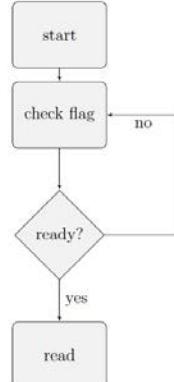
Interfaces in *.h Files	#include *.h Ort / Reihenfolge
<p>Functionality</p> <pre>void LED1_On(void); void LED1_Off(void); Void LED1_Neg(void); bool LED1_Get(void); void LED1_Put(bool); void LED_Init(void);</pre> <p>Initialization</p> <ul style="list-style-type: none"> Information Hiding! Eine XX_Init(void) Funktion sollte im Allgemeinen IMMER erstellt werden, auch wenn man sie vielleicht nicht braucht <p>Ein LED Driver für alle Plattformen: Anpassungen</p> <pre>#if PL_LED_CATHODE_PIN #define LED1_On() LED1_ClrVal() #else #define LED1_On() LED1_SetVal() #endif ... #if PL_LED_CATHODE_PIN #define LED_TURN_ON(nr) (LED##nr##_ClrVal()) #else #define LED_TURN_ON(nr) (LED##nr##_SetVal()) #endif #define LED1_On() (LED_TURN_ON(1)) #define LED2_On() (LED_TURN_ON(2))</pre> <ul style="list-style-type: none"> Obigen Code im LED.h einsetzen! Funktionsdeklarationen (Interfaces wie bspw. „void LED1_on(void)“ nicht vergessen! (siehe Interfaces in *.h Files) Dies bricht meiner Meinung nach mit der in Kapitel 3.1 genannten Konvention, ergibt aber effizienteren Code 	<p>Die Grafik rechts zeigt an welchen Ort die Header-Files inkludiert werden sollten.</p> <p>Die Reihenfolge sollte generell keine Rolle spielen. Tut es meistens aber doch:</p> <p>ACHTUNG:</p> <p>Ausschnitt aus Platform.h:</p> <pre>#define PL_IS_SRB_BOARD 1</pre> <p>Ausschnitt aus LED.h:</p> <pre>#if PL_IS_SRB_BOARD #define LED1_On() LED1_ClrVal() #endif</pre> <p>Ausschnitt aus Platform.c:</p> <pre>#include "LED.h" #include "Platform.h"</pre> <p>Fehler:</p> <p>PL_SRBOARD wird abgefragt, bevor es überhaupt definiert wurde. Zur Lösung des Problems die Reihenfolge der includes ändern:</p> <pre>#include "Platform.h" #include "LED.h"</pre>

4. Synchronization

4.1. Was ist Synchronisation? Wieso benötigt man Synchronisation?

- Synchronisation ist der Verbindungspunkt zwischen einzelnen Prozessen
- Computer arbeitet in unterschiedlichen Zeitskalen
 - Falls langsamer als reale Welt
 - Resultat zu spät: inkorrekte Resultat
 - Falls schneller als reale Welt
 - Resultat zu früh: inkorrekte Resultat
- Computer muss mit (realem Welt-Zeit-) Prozess synchronisieren
 - Beispiele: A/D-Converter; Keyboard
- **Computation Speed (Berechnungsgeschwindigkeit):**
 - Zu langsam: Problem
 - Zu schnell: Synchronisation

4.2. Arten von Synchronisation

Realtime	Gadfly („Polling“)
<p>Realtime Synchronisation benutzt einen Weg um die Programmausführung zu verzögern während sich das Device im State BUSY befindet. Diese Methode heisst Realtime weil es reale Zeit benutzt um zu warten (und dadurch zu synchronisieren). Die Verzögerung oder Wartezeit ist entweder von der Programmausführung abgeleitet (z.B. Dummy-Code um Zeit zu verbraten) oder benutzt explizite Funktionen (z.B. Wait10ms();).</p>	<p>Bei der Gadfly Synchronisation wird ständig ein Wert abgefragt (nicht wie beim Polling nur z.B. alle 10ms). Dies ist meistens irgendein Flag. Auf den Wert dieses Flags hin wird anschliessend eine Funktion ausgeführt oder eben nicht. Wird bei SCI oder AD/DA benutzt.</p>  <pre> graph TD start([start]) --> check[check flag] check -- no --> check check -- yes --> ready{ready?} ready -- yes --> read([read]) ready -- no --> check </pre>
<p>Probleme</p> <ul style="list-style-type: none"> • Ineffizient • Code overhead <ul style="list-style-type: none"> ○ Zusätzlicher Code ○ Zusätzliche Funktionsaufrufe • Abhängig vom Compiler • Abhängig vom Clock Speed <ul style="list-style-type: none"> ○ z.B. Wait10ms() wahrscheinlich nicht exakt 10ms lang 	<p>Vorteile</p> <ul style="list-style-type: none"> • Geringe Latenz <p>Probleme</p> <ul style="list-style-type: none"> • Benötigt viel Performance • Während dem Abfragen des Flags kann NICHTS anderes gemacht werden (ineffizient) • Benötigt z.B. ein Hardware-Flag
<p>Code Beispiel (mit expliziter Funktion)</p> <pre>void read(void) { uint8_t i; for(i = 0; i<sizeof(buffer); i++) { WaitMs(10); // Synchronisation Buffer[i] = PORTA; } }</pre>	<p>Code Beispiel</p> <pre>void read(void) { uint8_t i; for(i = 0; i<sizeof(buffer); i++) { while(PORTB.B0) // (raising edge) while(!PORTB.B0); // (falling edge) Buffer[i] = PORTA; } }</pre>

Interrupts	
<p>Realtime und Gadfly Synchronisation haben den grossen Nachteil, dass sie Prozessor-Zyklen (also Performance) benötigen während sie am Warten sind. Sind dies nur wenige Zyklen kann das akzeptabel sein, jedoch nicht bei längeren Wartezeiten.</p> <p>Die Idee hinter Interrupts ist, dass das System während dem Warten also etwas anderes machen kann. Es gibt folgende Arten von Interrupts:</p> <ul style="list-style-type: none"> • Software Interrupt • Hardware Interrupt <p>Software Interrupts sind ein bisschen tricky damit sie richtig funktionieren. Falls sie es nicht tun, ist der Fehler meistens nur schwer zu finden.</p>	<p>Ablauf einer ISR</p> <ul style="list-style-type: none"> • Die ISR muss so schlank wie möglich gehalten werden (kurze Ausführungszeit)! • Die Hauptarbeit wird von der „Main“ Funktion erledigt <p>Weitere Infos siehe Kapitel Interrupts.</p>
<p>Vorteile</p> <ul style="list-style-type: none"> • Während man wartet, kann anderer Code ausgeführt werden (bessere Performance) <p>Probleme</p> <ul style="list-style-type: none"> • Wichtig für Realtime-Systeme <ul style="list-style-type: none"> ◦ Interrupt Latency 	<p>Code Beispiel (wird im Allgemeinen vom ProcessorExpert gemacht)</p> <ul style="list-style-type: none"> • ISR Deklaration <pre>_interrupt void XX_Interrupt(void);</pre> • ISR Definition <pre>ISR(XX_Interrupt) { Clear XX_Interrupt Flag; /*Write User Code here;*/ }</pre>

Weitere Informationen zu Interrupts siehe nächstes Kapitel und LSC Interrupts.

4.2.1. Vergleich der Synchronisationsmethoden

	Effizienz	Hardware	Komplexität
Realtime	Schlecht / mittel	-	Kann sehr komplex sein
Gadfly	Schlecht, da Polling-Ansatz	-	Klein bis mittel
Interrupts	„Eventbasiert“, relativ gut	kann benötigt werden	Mittel, kann aber u.U. kompliziert sein

5. Interrupts

Ablauf einer Interrupt Service Routine (ISR)

- Die CPU wartet bis der aktuell ausgeführte Befehl beendet wird.
- Die CPU bestimmt die Adresse der auszuführenden ISR über ein Vektor-System.
- Die Rücksprungadresse wird auf den Stack gepusht.
- In vielen Prozessoren werden alle CPU Register auf den Stack gepusht.
- Das CPU Interrupt-Masken Bit wird gesetzt um weitere Interrupts zu unterbinden während der Laufzeit der ISR.
- CPU springt zur ISR.
- ISR wird abgearbeitet (möglichst kurz halten!)
- Die Rücksprungadresse wird vom Stack geholt
- Die CPU Register werden ebenfalls vom Stack geholt und entsprechend beschrieben.
- Allfällige weitere Interrupts werden „unmasked“

Interrupt-Maske

Kontroll-Bit welches entsprechende Interrupts stoppt, also nicht mehr von der CPU behandelt werden.

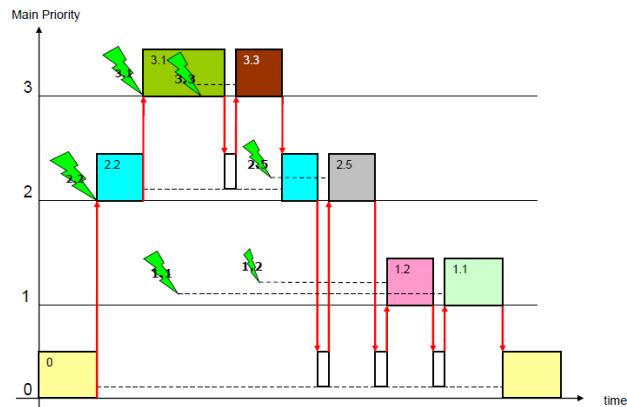
Interrupt-Latenz

Zeit zwischen Interrupt Bestätigung und Start der Ausführung der ISR. Gründe: Letzter Befehl wird noch fertig abgearbeitet, Register und Rücksprungadresse auf Stack pushen, andere ISR wird bereits ausgeführt

Prioritäten (Interrupt Rules)

Auch Interrupts können von einem anderen Interrupt unterbrochen werden. Um eine korrekte Ausführungsabfolge zu erreichen, kommen Prioritäten zum Einsatz. Der Interrupt mit der tieferen Hauptpriorität kann von einem Interrupt mit höherer Hauptpriorität unterbrochen werden. Evtl. sind auch Subprioritäten möglich (z.B. 1.2 im Bild nebenan).

- Priorität 0 kann je nach System/µP höchste aber auch tiefste Priorität (Base-Priority) sein.
- Bei nur zwei Prioritäten ist kein Nesting möglich.



Main-Funktion hat immer Base-Priority (hier 0)!

Vorgehen bei Implementation einer Interrupt Service Routine (ISR)

Allgemein

- Interrupt Vektor initialisieren (Vector Table)
- Entsprechende I/O's für Interrupt-Benutzung initialisieren (über Kontrollregister)
- Alle Interrupt Flags löschen, bevor die Interrupts unmasked werden (sonst könnte schon ein Interrupt ausgeführt werden, obwohl noch gar keiner auftrat).
- Die gewünschte Interrupt Quelle aktivieren (wird normalerweise im Interrupt-Device gemacht).
- Prozess-Loop: Die Applikation wird in einem Endlos-Loop ausgeführt

Was die ISR machen muss (Beispiel Implementations siehe vorherige Seite):

- Reset Interrupt-Flag
- Disable Lower Priority Interrupts: Falls der Interrupts mit höherer Priorität diesen Interrupt unterbrechen dürfen, disable lower priority Interrupts und unmask global interrupts
- ISR Funktion ausführen
- Re-enable Lower Priority Interrupts (falls sie vorher disabled wurden)
- Von der ISR zurückspringen

5.1. Design Hinweise für Interrupts

- Nicht zu viel verwenden!
- Nur für:
 - Dinge die nicht warten können
 - Reduktion von HW/SW Komplexität
 - Reduktion von Kosten
- ABER
 - Debugging ist ein Problem
 - Periodische und sporadische Fehler/Probleme möglich
- Anzahl BENÖTIGTE Interrupt-Quellen berücksichtigen
 - Intern, extern
 - Sicher gehen, dass nicht benötigte disabled/masked sind
- Sorgfältige Timing-Analyse
 - Unterbrechungen / Latenz
 - Abhängigkeiten
- Spezial Hardware verwenden

5.2. Reentrancy

Was ist Reentrancy?

Eine Funktion, Subroutine oder auch Interrupt kann jederzeit unterbrochen werden (durch Interrupts), ohne inkonsistente Daten oder Zustände zu verursachen.

Wieso Reentrancy?

- Eventuell greift eine ISR auf dieselbe Subroutine wie das Main Programm zu (Sharing Code)
- Eventuell greift eine ISR auf dieselben Daten wie das Main Programm zu (Sharing Data)
- Ein Interrupt kann zu JEDER Zeit stattfinden
- Konsequenz: Jede von einer ISR aufgerufene Subroutine muss reentrant sein!

5.2.1. Sharing Code & Sharing Data

Sharing Code (gemeinsame Subroutinen)	Sharing Data (gemeinsame Daten)
Vorteile: <ul style="list-style-type: none"> • Modularisation • Funktionalität 	Inkonsistente Zustände möglich <ul style="list-style-type: none"> • Datenzugriff muss geschützt werden! (z.B. mit Critical Sections)

6. Critical Sections

Wieso Critical Sections?

- Reentrancy

Wirkungsweise

Critical Sections können nicht von einem anderen Programmteil unterbrochen werden (auch nicht von Interrupts, da sie beim EnterCritical() ausgeschaltet werden). Dies führt dazu, dass der Code innerhalb der Critical Section garantiert in dieser Reihenfolge ausgeführt wird. Allerdings kann sich dadurch die Interrupt-Latenz erhöhen.

Funktionen

EnterCritical();

- Sichert den Inhalt der Status Register → SaveStatusReg()
- Disabled alle Interrupts

ExitCritical();

- Stellt Status Register wieder her → RestoreStatusReg
- Enabled alle Interrupts

Beispiel Implementation

```
uint8_t TRG_SetTrigger(TRG_TriggerKind trigger, TRG_TriggerTime ticks, TRG_Callback callback,  
TRG_CallBackDataPtr data) {  
    EnterCritical();  
    TRG_Triggers[trigger].ticks = ticks;  
    TRG_Triggers[trigger].callback = callback;  
    TRG_Triggers[trigger].data = data;  
    ExitCritical();  
    return ERR_OK;  
}
```

7. Doxygen

Ziel: Code ist gleichzeitig die Dokumentation (ähnlich wie bei javadoc)

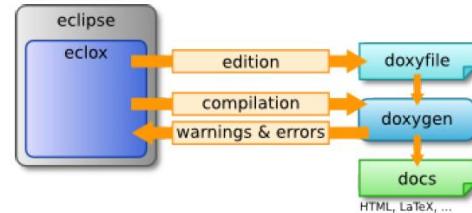
Dies führt zur Synchronisation von Code und Dokumentation.

Use cases für Doxygen

- Dokumentation
- System Überblick
- Listen wie z.B. To-Do Liste anfertigen

Um Doxygen zu nutzen wird folgendes benötigt:

- doxygen (Compiler Package)
- eclox (Eclipse Plugin)
- graphviz (Möglichkeit Grafiken zu visualisieren)

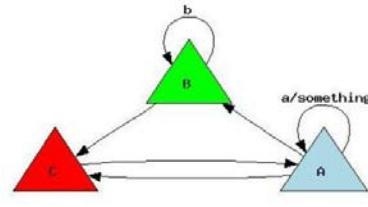


Beispiele Doxygen (alles Blaue gehört zum Doxygen Code)

Enumeration	
<pre>/*! Masks for LEDs, this is an enum, and the values are power of two to make it possible to build masks. The bit number also indicates the port bit number. Make sure compiler treats enum's efficient (e.g. as unsigned char, as enum by ANSI standard is int). */ typedef enum { LED_0 = (1<<0), /*! Bit0 of port for LED0 */ LED_1 = (1<<1), /*! Bit1 of port for LED1 */ } LED_Set;</pre>	<p>enum LED_Set</p> <p>Masks for LEDs, this is an enum, and the values are power of two to make it possible to build masks. The bit number also indicates the port bit number. Make sure compiler treats enums efficient (e.g. as unsigned char, as enum by ANSI standard is int).</p> <p>Enumerator:</p> <ul style="list-style-type: none"> <i>LED_0</i> Bit0 of port for LED0 <i>LED_1</i> Bit1 of port for LED1 <i>LED_2</i> Bit2 of port for LED2 <i>LED_3</i> Bit3 of port for LED3 <p>Definition at line 36 of file LED.h.</p>
Funktion und To-Do Liste für Projekt erstellen	
<pre>/* \brief Switches on a set of LED. \param[in] Leds The set of LED to be switched on. */ void LED_On(LED_Set Leds); void LED_On(LED_Set Leds) { /* Turns on all LED's ORED in the argument. */ /*!\todo implement function */ }</pre>	<p>Contents Index Search Favorites </p> <p>void LED_On (LED_Set Leds)</p> <p>Switches on a set of LED.</p> <p>Parameters: [in] Leds The set of LED to be switched on.</p> <p>Todo:implement function</p> <p>Definition at line 13 of file LED.c.</p> <p>File List</p> <ul style="list-style-type: none"> Lab File List <ul style="list-style-type: none"> derivative.h LED.c LED.h main.c platform.c platform.h Sta108.c Globals Related Pages
Module Header (Kopf einer Datei)	
<pre>/** * \file * \brief Key/Switch driver implementation. * \author Erich Styger, erich.styger@hslu.ch * \date 04.03.2011 * * This module implements a generic keyboard driver * for up to 4 Keys. * It is using macros for maximum flexibility with * minimal code overhead. */</pre>	<p>LED.c File Reference</p> <hr/> <p>Detailed Description</p> <p>LED driver.</p> <p>Author: Erich Styger, erich.styger@hslu.ch</p> <p>Here all the LED driver interfaces are defined.</p> <p>Definition in file LED.c.</p> <pre>#include "led.h" #include "derivative.h"</pre>

DOT Grafiken erstellen (zum Beispiel für State Machine Zustandsdiagramm; graphviz benötigt!)

```
\dot
digraph example_dot_graph {
node [shape=triangle]      // Form = Dreieck
rankdir=RL;      // von rechts nach links darstellen;
                  // R = rechts, L = Links, T = Top, D = Down
A [fillcolor=lightblue,style=filled,label="A" ]; // Farben und Label
B [fillcolor=green,style=filled,label="B" ];
C [fillcolor=red,style=filled,label="C" ];
A -> A [label="a/something"];      // Verbindungspeile mit Label
A -> B -> C -> A -> C;      // Verbindungspeile ohne Label
B -> B [label="b"];
}
\enddot
```



Bilder und Grafiken einfügen

Configuration File

```
/* project.doxygen */
IMAGE_PATH = my_image_path
```

Format des Codes:

```
\image <format> <file>
```

Code Beispiel

```
\image html my_image.jpg
```

8. State Machine (e.g. Mealy)

Der Zustandsautomat bzw. die Sequential State Machine (SSM) ist ein gebräuchliches Entwurfsmuster zur Realisierung ereignisgesteuerter Aktionsfolgen. Eine State Machine ist vergleichsweise einfach zu implementieren und bietet einen schematischen Weg um ein Problem zu lösen.

Eigenschaften einer State Machine

- Zustandsgesteuert
- Jeder Zustand sowie Zustandsübergang des Systems ist klar definiert
- Einfach zu implementieren
- Zustandsdiagramm kann mit Doxygen generiert werden (DOT Grafik, siehe Kapitel Doxygen)

Möglichkeiten, eine State Machine zu implementieren

- Konventionell mit normalen Funktionen (umständlich bei grösseren State Machines)
- Switch-Case Struktur (praktikabel wenn es nicht zu viele States und Übergänge gibt)
- Table Implementation (am meisten verdichteter Code)

Beispiel einer State Machine

<i>Switch-Case-Struktur</i>	<i>Table Implementation</i>									
<pre>typedef enum { A, B } StateT; static StateT state; void Loop(void) { for (;;) { r = Read(); switch(state) { case A: if (r == 1) { Write(c); } else { Write(a); State = B; } break; case B: if (r == 1) { Write(b); State = A; } else { Write(d); } break; } } }</pre>	<p><i>Table Implementation</i></p> <table border="1"> <thead> <tr> <th>State</th> <th>0</th> <th>1</th> </tr> </thead> <tbody> <tr> <td>A</td> <td>B/a</td> <td>A/c</td> </tr> <tr> <td>B</td> <td>B/d</td> <td>A/b</td> </tr> </tbody> </table> <pre>typedef enum { A, B } StateT; static StateT state; static uint8_t tbl[2][2][2] = {{{{B,a},{A,c}}}, {{{B,d},{A,b}}}}; void Loop(void) { uint8_t r; for(;;) { r = Read(); Output(tbl[state][r][1]); state = tbl[state][r][0]; } }</pre>	State	0	1	A	B/a	A/c	B	B/d	A/b
State	0	1								
A	B/a	A/c								
B	B/d	A/b								

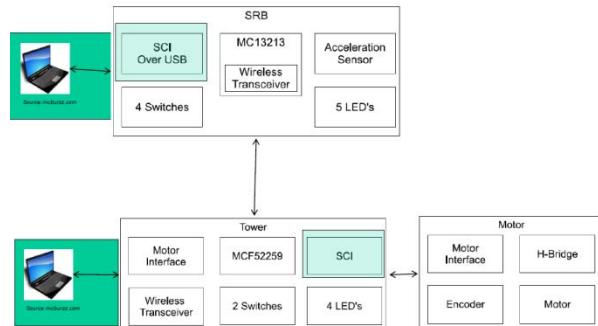
9. Shell

Was ist eine Shell?

Eine Shell (Schale, Hülle) ist ein User Interface oder eine Eingabe-Schnittstelle, welches dem User Funktionen zur Verfügung stellt. Diese kann jedoch auch für unterschiedliche Schnittstellen dieselben Funktionen zur Verfügung stellen.

Beispiel:

Im INTRO Projekt wird seriell über SCI (RS-232 über USB, SRB Board) und über USB (Tower Board) mit den Board kommuniziert. Mit der Shell können nun für beide Schnittstellen dieselben Funktionen benutzt werden, obwohl die physische Auslegung verschieden ist. Die Shell stellt die Funktionen zur Verfügung.



Von Shell bereitgestellte Funktionen (Auszug, Befehle können z.B. mit HTerm übermittelt werden)

- Hilfe-Texte ausgeben
 - Status ausgeben
 - Befehl senden

9.1. Implementation einer Shell

Shell Interface

```
/*!
 * \brief Passes a command line string to the shell parser
 * \param cmd string to be parsed
 */
void SHELL_ParseLine(const char_t *cmd);
/*! \brief Reads input (if any) from the console and processes it */
void SHELL_ReadAndParseLine(void);
/*!
 * \brief sends a message to the console
 * \param msg pointer to the string to send.
 */
void SHELL_SendMessage(const char *msg);
/*! \brief Serial driver initialization */
void SHELL_Init(void);
```

Shell Funktionen für individuelle Module einfügen/benutzen (hier für Mealy State Machine)

Aus Mealy.c

```
/*!
 * \brief Prints the status text to the console
 * \param io I/O channel to be used
 */
static void PrintStatus(const FSSH1_StdIOType *io) {
    FSSH1_SendStatusStr("Mealy Status", MEALY_isOn?"on\r\n":"off\r\n", io->stdOut);
}
```

```
/*
 * \brief Prints the help text to the console
 * \param io I/O channel to be used
 */
static void PrintHelp(const FSSH1_StdIOType *io) {
    /* list your local help here */
    FSSH1_SendHelpStr("mealy", "Group of Mealy commands\r\n", io->stdOut);
    FSSH1_SendHelpStr(" help|status", "Print help or status information\r\n", io->stdOut);
    FSSH1_SendHelpStr(" on|off", "Turns the machine on or off\r\n", io->stdOut);
}

/*
 * \brief Übersetzt die eingegangenen Befehle und ruft die entsprechende Funktion auf
 */
uint8_t MEALY_ParseCommand(const char *cmd, bool *handled, const FSSH1_StdIOType *io) {
    if (UTIL1_strcmp(cmd, FSSH1_CMD_STATUS)==0 || UTIL1_strcmp(cmd, "mealy status")==0) {
        PrintStatus(io);           // eingegebener Befehl über HTerm war "mealy status" (ohne „)
        *handled = TRUE;
    } else if (UTIL1_strcmp(cmd, FSSH1_CMD_HELP)==0 || UTIL1_strcmp(cmd, "mealy help")==0) {
        PrintHelp(io);            // eingegebener Befehl über HTerm war "mealy help" (ohne „)
        *handled = TRUE;
    } else if (UTIL1_strncmp(cmd, "mealy on", sizeof("mealy on")-1)==0) {
        MEALY_isOn = TRUE;        // eingegebener Befehl über HTerm war "mealy on" (ohne „)
        *handled = TRUE;
    } else if (UTIL1_strncmp(cmd, "mealy off", sizeof("mealy off")-1)==0) {
        MEALY_isOn = FALSE;       // eingegebener Befehl über HTerm war "mealy off" (ohne „)
        *handled = TRUE;
    }
    return ERR_OK;
}
```

Aus Shell.c

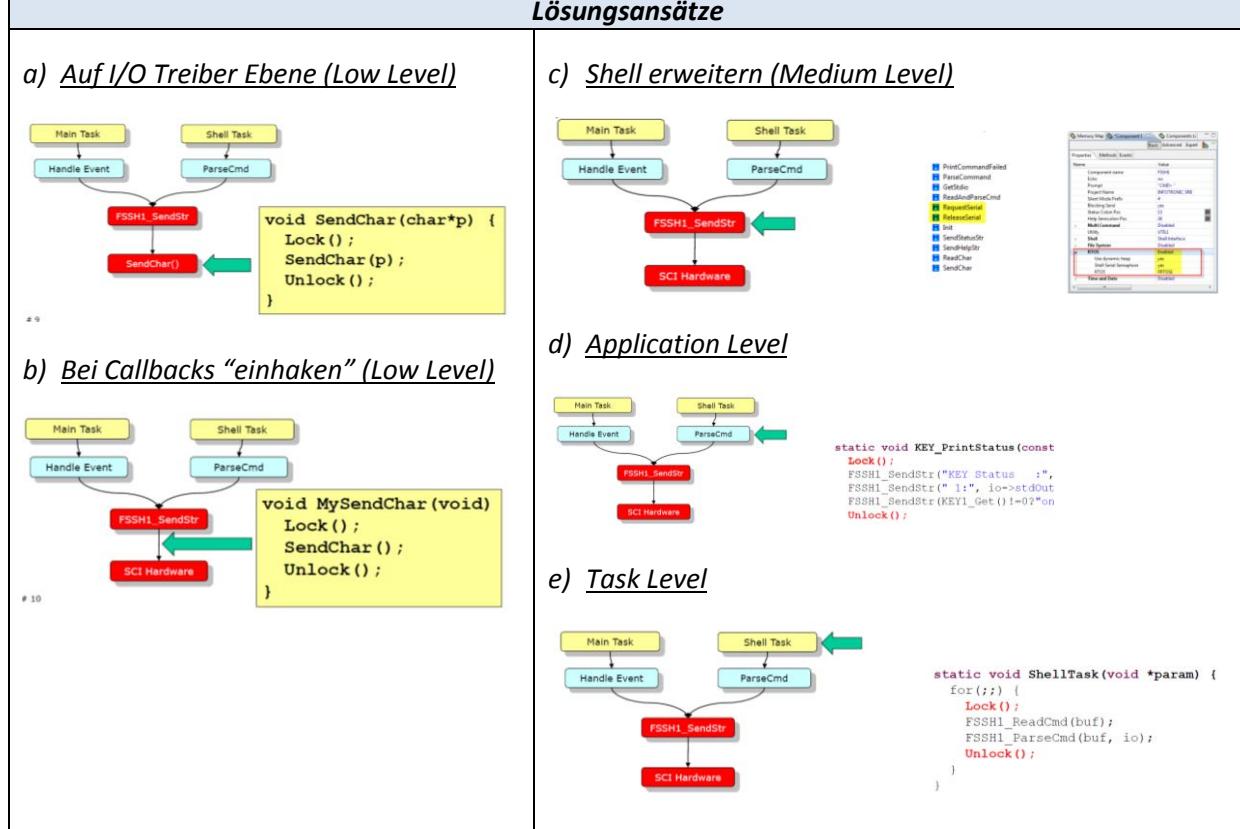
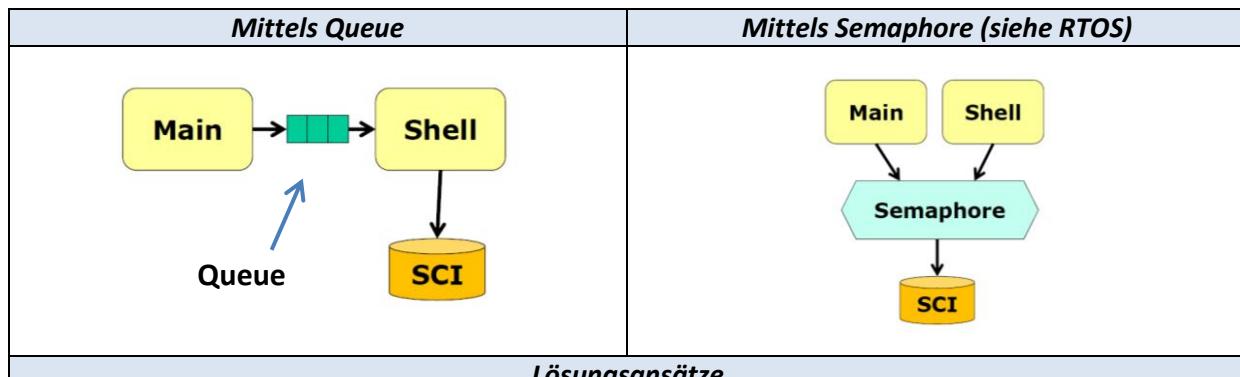
```
/*
 * \brief Übersetzt die eingegangenen Befehle und ruft die entsprechende Funktion auf
 */
static uint8_t ParseCommand(const unsigned char *cmd, bool *handled, const FSSH1_StdIOType *io) {
    /* handling our own commands (Shell.c own commands) */
    if (UTIL1_strcmp(cmd, FSSH1_CMD_HELP)==0) {
        PrintHelp(io);           // ruft die Help-Ausgabe von Shell.c auf
        *handled = TRUE;
    } else if (UTIL1_strcmp(cmd, FSSH1_CMD_STATUS)==0) {
        PrintStatus(io);         // ruft die Status-Ausgabe von Shell.c auf
        *handled = TRUE;
    }
    /*!\Handle here the additional parsers, e.g. Mealy in this example */
#ifndef PL_HAS_MEALY
    if (MEALY_ParseCommand(cmd, handled, io)!=ERR_OK) {
        return ERR_FAILED;
    }
    return ERR_OK;             // nicht vergessen, den Return-Wert zurückzugeben
}
```

9.2. Shell Protected Access

Beim geschützten Zugriff geht es darum, dass nur von einer Quelle gleichzeitig auf die serielle Schnittstelle (SCI) zugegriffen wird. Ansonsten könnte es dazu kommen, dass von unterschiedlichen Quellen z.B. gleichzeitig Daten geschickt werden und somit keine korrekte Datenübertragung garantiert werden kann.

Beispiel	
Quelle 1	Quelle 2
Daten zur Übertragung = „Hallo“	Daten zur Übertragung = „Welt“
Beide fangen nun gleichzeitig an zu senden Ergebnis Ausgabedaten = „H W a e l l t o“ → Daten werden miteinander vermischt!	

9.2.1. Zugriff auf gemeinsame Ressourcen



10. Simple Events

Ein Schlüsselement eines Realtime Systems ist die Interaktion mit der realen Welt. Um dies zu erreichen, bieten sich Events sehr gut an. Die Applikation macht solange nichts, bis ein Event (z.B. Interrupt) auftritt.

Event System	
Allgemein	INTRO
<pre> graph TD ISR[ISR] -- "SetEvent()" --> eventArray[event array] eventArray --> eventLoop[event loop] eventLoop -- "GetEvent()" --> ISR </pre>	<pre> graph TD KBI_ISR[KBI ISR] --> Keys[Keys] Keys --> Debounce[Debounce] Keys --> APP_MainLoop[Application Main Loop] APP_MainLoop -- "KEY1_GetVal(), KEY_Poll()" --> Keys APP_MainLoop -- "EVNT_SetEvent()" --> Debounce Debounce --> Event[Event] Event -- "EVNT_SetEvent()" --> APP_MainLoop Event -- "EVNT_HandleEvent()" --> EventList[Event List] </pre>
Interrupts und Events	Speichern von Events
<pre> graph LR subgraph fast [fast] SRB[SRB] --> ISR[ISR] ISR --> Poll[Poll] Poll --> EventLoop[Eventloop] end subgraph slow [slow] Tower[Tower] --> EVNT_SetEvent[EVNT_SetEvent] EVNT_SetEvent --> EventLoop end </pre>	Mittels #define <pre> #define EVNT_INIT 0 /*!< Initialization Event */ #define EVNT_SW1_PRESSED 1 /*!< SW1 pressed */ #define EVNT_SW2_PRESSED 2 /*!< SW2 pressed */ </pre> Mittels enum <pre> typedef enum EVNT_Handle { EVNT_INIT, /*!< System Initialization Event */ EVNT_SW1_PRESSED, /*!< SW1 pressed */ EVNT_SW2_PRESSED, /*!< SW2 pressed */ ... } EVNT_Handle; </pre>
Event Implementation	
SetEvent() und Clear Event()	EventHandle()
<pre> void EVNT_SetEvent(EVNT_Handle event) { EnterCritical(); SET_EVENT(event); ExitCritical(); } void EVNT_ClearEvent(EVNT_Handle event) { EnterCritical(); CLR_EVENT(event); ExitCritical(); } </pre>	<pre> void EVNT_HandleEvent(void (*callback)(EVNT_Handle)) { /* Handle the one with the highest priority. Zero is the event with the highest priority. */ EVNT_Handle event; EnterCritical(); for (event=(EVNT_Handle)0; event<EVNT_NOF_EVENTS; event++) { /* does a test on every event */ if (GET_EVENT(event)) { /* event present? */ CLR_EVENT(event); /* clear event */ break; /* get out of loop */ } } ExitCritical(); if (event != EVNT_NOF_EVENTS) { callback(event); } } </pre>
Event-Handling auf Application Ebene	
<pre> graph TD APP_Main[APP_Main()] --> EVNT_HandleEvent[EVNT_HandleEvent] EVNT_HandleEvent --> APP_HandleEvent[APP_HandleEvent] APP_HandleEvent -- "switch(event)" --> EVNT_HandleEvent EVNT_HandleEvent -- "EVNT_INIT" --> Benachrichtigung[Benachrichtigung] Benachrichtigung --> APP_HandleEvent APP_HandleEvent --> EVNT_HandleEvent EVNT_HandleEvent -- "Iterate through Events" --> APP_HandleEvent APP_HandleEvent -- "If (EVENT_EventSet()) {" --> EVNT_HandleEvent EVNT_HandleEvent -- "EVNT_ClearEvent();" --> APP_HandleEvent APP_HandleEvent -- "If (there is event) {" --> EVNT_HandleEvent EVNT_HandleEvent -- "callback(event);" --> APP_HandleEvent </pre>	

11. Clock and Timer

11.1. Clock

In einem Mikrocontrollersystem sind immer mehrere Clocks vorhanden:

- System Clock
- CPU Clock
- Bus Clock
- Evtl. weitere

Je nach System können diese Clocks gleich oder unterschiedlich sein (häufig ist z.B. der Bus Clock niedriger als der CPU Clock). Die Clock-Source/Prescaler etc. werden im ProcessorExpert ausgewählt.

11.2. Timer

Mit einem Timer kann ein periodischer Ablauf gesteuert werden. Für Real-Time Systeme ist dies eminent wichtig (periodische Ticks wie z.B. Prozessausführung alle 10ms). Real-Time Systeme benötigen also:

- Zeitbasis
- Clock
- Interrupt Synchronisation

Dies wird von Timern abgeleitet. Dieser kann mit externem oder internem Clock/Oscillator betrieben werden.

Durch Timer erhalten wir folgende Funktionen/Eigenschaften	
• Verbindung zu realer Zeit (ns, µs, ms, s, h, ...)	• Periodische Ticks <ul style="list-style-type: none">○ Zeit der Tick-Periode bekannt; externe oder interne Quelle; System/CPU/Bus Clock
Vorgehen bei Implementation eines Timers (ProcessorExpert)	
• Auswahl des Referenz-Clocks (intern oder extern)	• Auswahl des Prescalers (Faktor zwischen Referenz-Clock und Timer-Clock) → Periodendauer
Beispiel Implementation	
Interface <pre>/* we get called every 10 ms */ #define TMR_TICK_MS 10 /* * \brief Function called from timer interrupt * every TMR_TICK_MS. */ void TMR_On10ms(void); /*! \brief Timer driver initialization */ void TMR_Init(void);</pre>	Implementation <pre>void TMR_On10ms(void) { /* timer interrupt is calling us every 10 ms */ #if PL_HAS_LED_HEARTBEAT /* we are using a timer to do the heartbeat */ static uint8_t cnt = 0; /* using static local variable */ cnt++; if (cnt==1000/TMR_TICK_MS) { /* every second */ EVNT_SetEvent(EVNT_LED_HEARTBEAT); /* using event method */ } #endif cnt = 0; /* reset counter */ } }</pre>

12. Simple Trigger

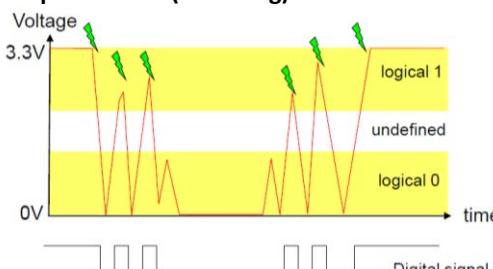
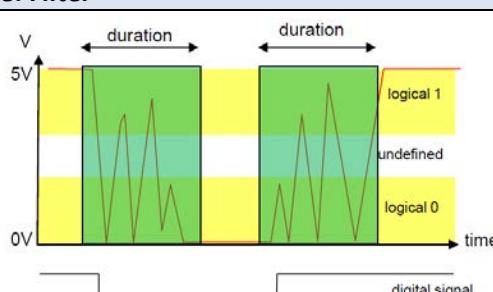
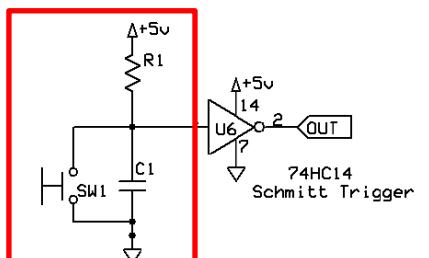
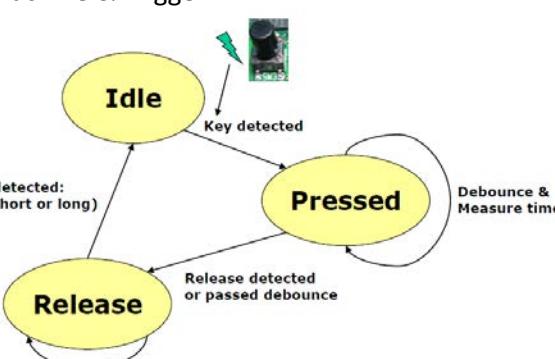
Mit einem Trigger lassen sich zukünftige Events auslösen (triggern). Beispiele Trigger Benutzung:

- Synchronisation
- LED alle 500ms leuchten lassen (oder mehrere mit unterschiedlichen Periodendauern)

Design Idee	
<ul style="list-style-type: none"> • Infrastruktur um Dinge periodisch auszuführen • „fire and forget“: z.B. mach das in 400ms und man muss sich nicht mehr darum kümmern • Geeignet für eher kleine Dinge wie eine LED blinken lassen • Möglichst wenige Ressourcen des Mikrocontrollers benötigen <ul style="list-style-type: none"> ◦ Minimale Speichernutzung ◦ Nur 1 Timer → reuse ◦ Universelles Interface • Sollte nicht zu viele Aufgaben übernehmen (höchstens etwas mehr als 10) • Einfach zu benutzen und zu verstehen • Sollte mit und ohne RTOS funktionieren 	
Funktionsweise	Trigger Interface
<p>ISR ist hier z.B. TimerOn10ms()</p>	<pre>/* * \brief Adds a new trigger * \param trigger Trigger to be added * \param ticks Trigger time in ticks. The time is relative from the * current time. * \param callback Callback to be called when the trigger fires * \param data Optional pointer to data * \return error code, ERR_OK if everything is fine */ uint8_t TRG_SetTrigger(TRG_TriggerKind trigger, TRG_TriggerTime ticks, TRG_Callback callback, TRG_CallBackDataPtr data);</pre> <pre>/*!\brief Called from interrupt service routine with a period of TRG_TICKS_MS.*/ void TRG_IncTick(void);</pre> <pre>/*!\brief Initializes the module. */ void TRG_Init(void);</pre>
Trigger Descriptor (Datenstruktur eines Triggers)	Vom System benutzbare Trigger definieren (static data allocation)
<pre>/*!\brief Descriptor for a trigger.*/ typedef struct TRG_TriggerDesc { TRG_TriggerTime ticks; /*!< tick count until trigger */ TRG_Callback callback; /*!< callback function */ TRG_CallBackDataPtr data; /*!< additional data pointer for callback */ } TRG_TriggerDesc;</pre>	<pre>/*!\brief Triggers which can be used from the application*/ typedef enum { /*!\Extend the list of triggers as needed */ TRG_LED_BLINK, /*!< LED blinking */ TRG_BTNLED_OFF, /*!< Turn LED off */ TRG_BTNSND_OFF, /*!< Switch sounder off */ TRG_KEYPRESS, /*!< Trigger for debouncing */ TRG_NOF_TRIGGER /*!< Must be last! */ } TRG_TriggerKind;</pre>

Reentrancy: Trigger reentrant machen (rot markiert)	
<pre> void TRG_IncTick(void) { TRG_TriggerKind i; EnterCritical(); for(i=(TRG_TriggerKind)0;i<TRG_NOF_TRIGGER;i++) { if (TRG_Triggers[i].ticks!=0) { /* prevent underflow */ TRG_Triggers[i].ticks--; } } /* for */ ExitCritical(); while(CheckCallbacks()) {} /* while we have callbacks, re-iterate the list as this may have added new triggers at the current time */ } </pre>	<pre> static bool CheckCallbacks(void) { TRG_TriggerKind i; TRG_Callback callback; TRG_CallBackDataPtr data; bool calledCallBack = FALSE; for(i=(TRG_TriggerKind)0;i<TRG_NOF_TRIGGER;i++) { EnterCritical(); if (TRG_Triggers[i].ticks==0 && TRG_Triggers[i].callback != NULL) { /* trigger! */ callback = TRG_Triggers[i].callback; /* get a copy */ } data = TRG_Triggers[i].data; TRG_Triggers[i].callback = NULL; ExitCritical(); callback(data); calledCallBack = TRUE; } else { ExitCritical(); ← } } /* for */ return calledCallBack; } </pre> <p style="color: red;">ACHTUNG! Muss wegen dem if-else auch hier stehen!</p>
Callbacks	
<p>Callback nennt sich die Funktion, welche vom Trigger aufgerufen wird. Dies ist meistens dieselbe Funktion von wo SetTrigger ausgeführt wird:</p> <pre> #if PL_HAS_TRIGGER static void LedBlink(void *p) { (void)p; /* avoid compiler warning */ LED2_Neg(); (void)TRG_SetTrigger(TRG_LED_BLINK, 500/TRG_TICKS_MS, LedBlink, 0); } </pre>	<p style="text-align: right;">Callback Funktion</p>
Relative Time Triggers: Delay / Accuracy; Ausführung eines Triggers	Daten mithilfe eines Triggers übergeben → Pointer
<ul style="list-style-type: none"> - Action for the future - Relative, delta to current time (#ticks) - Simplicity vs. Timer Resolution vs. Accuracy 	<pre> uint8_t TRG_SetTrigger(TRG_TriggerKind trigger, TRG_TriggerTime ticks, TRG_Callback callback, TRG_CallBackDataPtr data) { EnterCritical(); TRG_Triggers[trigger].ticks = ticks; TRG_Triggers[trigger].callback = callback; TRG_Triggers[trigger].data = data; ExitCritical(); return ERR_OK; } TRG_SetTrigger(TRG_BTNLED_OFF, 3, BlinkLED, NULL); </pre>

13. Debounce

Wieso Debounce?	
Durch das Prellen (Bouncing) eines jeden Tasters kann es sein, dass anstatt nur ein einziger Interrupt mehrere Interrupts ausgelöst werden. Durch das Debouncing oder Entprellen kann dem entgegengewirkt werden.	Beispiel Prellen (Bouncing) 
Design Idee: Filter	
Während der Zeit des Prellens wird der Input nicht mehr abgefragt (Software) oder die Signal-Peaks eliminiert (Hardware). Die Filterzeit (duration) muss empirisch durch Messen ermittelt werden (kann NICHT im vornherein genau bekannt sein!).	
Hardware	Software
<p>Pull-Up Widerstand und Kondensator</p>  <p>Choose RC > duration of bounce, in seconds</p>	<p>State Machine & Trigger</p>  <p>Da über einen Interrupt in die State Machine gesprungen wurde und man dort nicht bleiben kann (andere Sachen sollen ja auch noch ausgeführt werden), wird mit einem Trigger in die State Machine zurückgesprungen. Jeder Zustandsübergang wird somit getriggert.</p> <p>ACHTUNG 1: Das ganze muss reentrant sein (z.B. wegen mehreren Switches aber selben State Machine).</p> <p>ACHTUNG 2: Das setzen des Triggers erfolgt infolge der Befehlsabarbeitung erst einige Zeit/Zyklen nach dem Auslösen des Interrupts → Timing</p>

14. RTOS / FreeRTOS

14.1. RTOS

RTOS Anforderungen						
<ul style="list-style-type: none"> • Vorhersehbarkeit <ul style="list-style-type: none"> ◦ Deterministisch, Services mit einer bekannten maximalen Ausführungszeit • Präzises Timing <ul style="list-style-type: none"> ◦ Timing und Scheduling, Timer • Geschwindigkeit <ul style="list-style-type: none"> ◦ Computing Power für die Applikation 						
Wieso wird ein RTOS benutzt?						
Genau wie beim PC: Es stellt Services und Funktionen zur Verfügung						
OS Services						
<ul style="list-style-type: none"> • Bündelung von Ressourcen • Prozesse <ul style="list-style-type: none"> ◦ Scheduling, Switching, Synchronisation, Life Cycle • HW Ressourcen <ul style="list-style-type: none"> ◦ HW Abstraktion (HAL), Zugriffskontrolle, Real-Time Clock • Speicher <ul style="list-style-type: none"> ◦ Heap, MMU (Memory Management Unit) • Dateisystem <ul style="list-style-type: none"> ◦ File I/O • Kommunikation <ul style="list-style-type: none"> ◦ GUI (HMI), Protocol Stacks (M2M), Inter-Prozess-Kommunikation (IPC) 						
<pre> graph TD Application[Application] <--> API[API] API <--> RTOS[RTOS] RTOS <--> HAL[HAL] HAL <--> Hardware[Hardware] </pre>						
Embedded Anforderungen						
<ul style="list-style-type: none"> • Sehr breit gefächert • Kritische Applikationen <ul style="list-style-type: none"> ◦ Hohe Funktionalität (Medizin, Weltraum Erforschung, ...) ◦ Niedrige Funktionalität (ABS, Tempomat, ...) • Unkritische Applikationen <ul style="list-style-type: none"> ◦ Hohe Funktionalität (PDA, Smartphones, ...) ◦ Niedrige Funktionalität (SmartCard, Ofen, ...) 						
Standard OS vs. Embedded OS						
<table border="1" style="width: 100%; border-collapse: collapse;"> <tr> <td style="width: 33%; padding: 5px;"> Standard OS </td> <td style="width: 33%; padding: 5px;"> RTOS </td> <td style="width: 33%; padding: 5px;"></td> </tr> <tr> <td style="padding: 5px;"> Application Middleware Middleware Operating System Device Driver Device Driver Hardware </td> <td style="padding: 5px;"> Application Middleware Middleware Device Driver Device Driver Real Time Kernel Hardware </td> <td style="padding: 5px;"> Drivers: Standard = Teil des OS Embedded = Tasks </td> </tr> </table>	Standard OS	RTOS		Application Middleware Middleware Operating System Device Driver Device Driver Hardware	Application Middleware Middleware Device Driver Device Driver Real Time Kernel Hardware	Drivers: Standard = Teil des OS Embedded = Tasks
Standard OS	RTOS					
Application Middleware Middleware Operating System Device Driver Device Driver Hardware	Application Middleware Middleware Device Driver Device Driver Real Time Kernel Hardware	Drivers: Standard = Teil des OS Embedded = Tasks				

RTOS Kernel Architecture	
<pre> graph LR Interrupt --> ID[Interrupt Dispatcher] TimerInterrupt --> TSE[Time Service & Events] SystemCall[System Call (Traps)] --> OS[OS Services] ID --> SDD[Scheduler & Dispatcher] TSE --> SDD OS --> SDD SDD --> Task[Task] </pre>	<ul style="list-style-type: none"> • Interrupt Dispatcher <ul style="list-style-type: none"> ◦ Erkennt welcher Interrupt ausgelöst wurde und ruft entsprechende Interrupt Service auf • Interrupt Service <ul style="list-style-type: none"> ◦ Interrupt Service Routine (ISR) • Time Service & Events <ul style="list-style-type: none"> ◦ Timer Interrupts (von normalen Timern, RTOS Ticks etc.) • OS Services <ul style="list-style-type: none"> ◦ Siehe „OS Services“ • Scheduler & Dispatcher <ul style="list-style-type: none"> ◦ Regelt die Prozessausführung; d.h. welcher Prozess wird jetzt ausgeführt und welcher später
Scheduling & Context Switch (Prozesswechsel)	
<pre> graph LR Clock((Clock)) -- Tick --> Scheduler[Scheduler] Scheduler -- Wait --> Task1((Task)) Scheduler -- Yield --> Task2((Task)) Scheduler -- Sync --> Task3((Task)) </pre> <p>Preeemptive Non-Preeemptive</p>	<p><u>No-Preemptive (nicht unterbrechend)</u> Sobald dem Prozess die CPU zugeteilt wurde, wird der Prozess so lange laufen, bis er von selbst oder wenn er blockiert die CPU wieder freigibt.</p> <p><u>Preeemptive (unterbrechend)</u> Immer den höchsten verfügbaren Task ausführen. Tasks mit identischer Priorität teilen CPU-Zeit.</p>
OS Process States	
<pre> graph TD Run((Run)) -- terminate --> End(()) Run -- wait --> Wait((Wait)) Wait -- signal --> Ready((Ready)) Ready -- dispatch --> Run Ready -- preempt --> Idle((Idle)) Idle -- resume --> Ready Idle -- end cycle --> Run Ready -- activate --> Start(()) </pre>	<p>Das RTOS hat nur eine minimale Anzahl States (Komplexität des RTOS davon abhängig)</p>
Auswahlkriterien für ein spezifisches RTOS	
<ul style="list-style-type: none"> • Funktionalität (API, Stacks, Standards, ...) • Programmiersprachen • Real-time oder nicht? • Benötigte Ressourcen • Skalierbarkeit • Verfügbarkeit • Offenheit • Tools & Support • Kosten- & Lizenzierungsmodell • Statisch oder dynamisch? • Eigene Vorlieben 	<p><u>Nichtfunktionale Faktoren</u></p> <ul style="list-style-type: none"> • Lernkurve • Synergien innerhalb der Firma • Informationen & Dokumentation • Qualität (Produkt, Dienstleister, ...) • Zertifikation (SIL, DOI, OSEK, ...) • Lizenzierungsmodell • Kosten für Evaluation, Einführung und Unterhalt <p>→ Werden oftmals vergessen!</p>

14.2. FreeRTOS

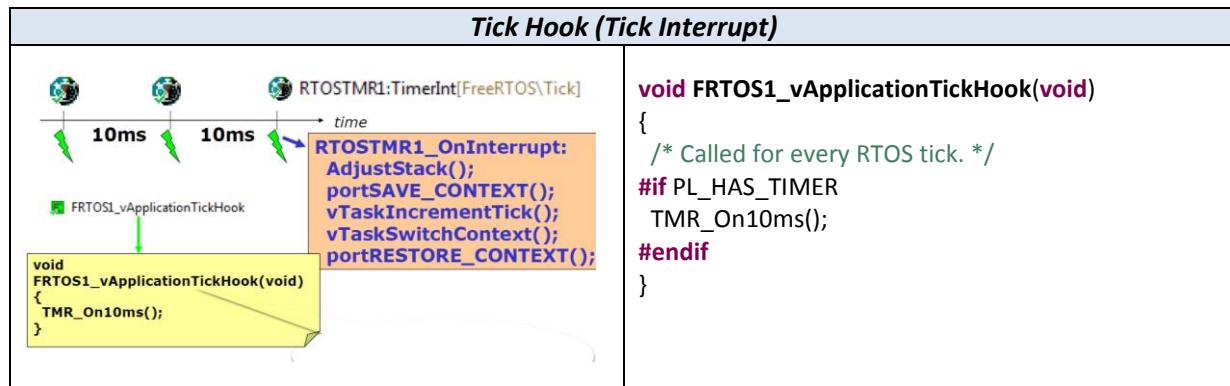
Stichworte zu FreeRTOS

- Einfach, portierbar, lizenzfrei, kurzgefasst (nicht überladen)
- Micro Real-time Kernel
- OS hat Task mit niedrigster Priorität
- Wahl des RTOS Scheduling Grundsatzes:
 - **Pre-emptive:** Immer den höchsten verfügbaren Task ausführen. Tasks mit identischer Priorität teilen CPU-Zeit
 - **Cooperative:** Context switches treten nur auf wenn ein Task blockiert oder explizit anfordert
- Message Queue
- Semaphoren (via Makros)
- RTOS Kernel nutzt mehrere Prioritätslisten
- Ports für diverse Mikrocontroller verfügbar (TI MSP430, Atmel AVR, Freescale HCS12 & Coldfire, etc.)

RTOS Ressourcen	
<ul style="list-style-type: none"> • SWI Interrupt <ul style="list-style-type: none"> ◦ Periodic tick interrupt (timer) ◦ Preemption ◦ Context switch ◦ Time base • Memory/Stack <ul style="list-style-type: none"> ◦ Tasks ◦ Queues ◦ Messages • Configuration <ul style="list-style-type: none"> ◦ FreeRTOSConfig.h ◦ Tick Rate (Hz) ◦ Heap Size ◦ Minimal Stack Size 	
FreeRTOS Processor Expert Komponente Features	
<ul style="list-style-type: none"> • 1ms Performance Counter • TraceHooks • Software Interrupt • Tick Counter (z.B. 10ms für 100Hz) 	<p><u>Für Shell Komponente (FSSHx)</u></p> <ul style="list-style-type: none"> • RTOS → enabled
System Startup	
<p>Das RTOS wird im APP_Init() aufgerufen und somit die Tasks generiert, welche anschliessend ausgeführt werden.</p> <p><u>Hinweis:</u> Ein Idle-Task wird IMMER erstellt, auch wenn keine anderen Tasks erstellt werden.</p>	<pre> graph TD _Startup_ --> main[main()] main --> APP_Init[APP_Init()] APP_Init --> RTOS_Run[RTOS_Run()] RTOS_Run --> CreateTasks[Create Tasks; vTaskStartScheduler()] CreateTasks -.-> FreeKernelMemory((Free Kernel Memory)) CreateTasks --> InitList["Create Idle Task
Enable Tick Timer
Setup 1st task context
RTI"] </pre>

14.2.1. Ticks

Durch Ticks wird eine periodische Ausführung des Codes realisiert. Diese kann bei der Processor Expert Komponente eingestellt werden (z.B. 100Hz ergeben eine 10ms Periodendauer).



14.2.2. Speicher allozieren (für Tasks, Queues und Semaphoren)

FreeRTOS Memory Schemes	
Scheme 1	Alloziert nur Speicher. Kein vTaskDelete(), vQueueDelete(), ...
Scheme 2	Speicherblock kann freigegeben werden. Verbindet freie Speicherblöcke nicht → Fragmentierung möglich! Ungünstig für zufällige alloc/free Sequenzen
Scheme 3	Scheme für standard malloc() und free()
Scheme 4	Verbindet freie Speicherblöcke
Malloc(), Free() und FreeHeap()	
<pre> void *pvPortMalloc(size_t xWantedSize); void vPortFree(void *pv); size_t xPortGetFreeHeapSize(void); </pre>	<u>Beispiel</u> <pre> bufP = (char_t*)pvPortMalloc(sizeof("Hello")); vPortFree(bufP); </pre>

14.2.3. Tasks

Tasks erledigen die eigentliche Arbeit des Mikrocontroller-Programmes.

Task Creation (xTaskCreate(), xTaskDelete())	Create Task and start FreeRTOS Scheduler
<pre> xTaskCreate(MyTask, (signed char*)"MyTask", configMINIMAL_STACK_SIZE, (void*)myParam, tskIDLE_PRIORITY, &myTaskHandle); static portTASK_FUNCTION(MyTask, pvParameters) { (void)pvParameters; for(;;) { EVNT_HandleEvents(); } /* loop forever */ } </pre> <p><u>ACHTUNG:</u> Die Minimal Stak Size wird in Elementen gerechnet, NICHT in Bytes! D.h. bei 32-Bit CPU und Stack Size = 200 → 200*(4*Bytes) = 800 Bytes auf Stack</p>	<pre> void RTOS_Run(void) { if (FRTOS1_xTaskCreate(MainTask, // Name des Tasks (signed portCHAR *)"Main", configMINIMAL_STACK_SIZE+100, (void*)NULL, tskIDLE_PRIORITY, (xTaskHandle*)NULL) != pdPASS) { for(;;){ /* error! probably out of memory */ } FRTOS1_vTaskStartScheduler(); } </pre>

Task Switches passieren auf...	
<ul style="list-style-type: none"> • Tick Interrupt • taskYIELD • RTOS API call 	
FreeRTOS API	
vTaskDelay(#ofTicks) Der Task wartet nach der Ausführung so viele Ticks bis zur nächsten Ausführung.	<pre>void vTaskFunction(void *pvParameters) { for(;;) { /* toggle the LED every 500ms */ LED0_Neg(); EVNT_HandleEvent(APP_HandleEvent); vTaskDelay(500/portTICK_RATE_MS); } /* for */ }</pre>
<i>Weitere Funktionen (S. 5-14 Folien SW6e):</i> <ul style="list-style-type: none"> • vTaskStartScheduler(), vTaskEndScheduler() • uxTaskPriorityGet(), vTaskPrioritySet() • vTaskSuspendAll(), vTaskSuspend() • vTaskResumeAll(), vTaskResume(), vTaskResumeFromISR() • vTaskYIELD() • void taskENABLE/DISABLE_INTERRUPTS(void) • void taskENTER/EXIT_CRITICAL(void) 	<i>Erläuterungen:</i> <ul style="list-style-type: none"> • Scheduler starten / stoppen • Task Priorität modifizieren • Task unterbrechen • Zu Task zurückkehren und ausführen • Forciert Context Switch • Globale Interrupt Kontrolle • Für „atomare“ Prozesse (siehe Kap. 6)
Idle Tasks	Task Transition Diagramm
<u>Pseudo Code für Idle Task</u> <pre>static void prvIdleTask(void *pvParameters) { for(;;) { RemoveDeletedTasksFromList(); if (!configUSE_PREEMPTION) { taskYIELD(); } else if (configIDLE_SHOULD_YIELD) { if (NofReadyTasks(tskIDLE_PRIORITY)>1) { taskYIELD(); } } IdleHook(); } /* for */ }</pre>	<pre> graph TD S[Suspended] -- "vTaskSuspend() called" --> R[Ready] S -- "vTaskSuspend() called" --> B[Blocked] R -- "vTaskResume() called" --> R R -- "vTaskResume() called" --> B B -- "vTaskResume() called" --> R B -- "vTaskResume() called" --> R B -- "Blocking API function called" --> B E[Event] --> R </pre>
Idle Should Yield Konfiguration (I = Idle Task)	
- configIDLE_SHOULD_YIELD set to 0	- configIDLE_SHOULD_YIELD set to 1

14.2.4. Semaphore & Mutex

Semaphoren lösen das Problem beim Benutzen von gemeinsamen Daten / Ressourcen, dass nicht gleichzeitig darauf zugegriffen werden kann / darf → Critical Section & Synchronisation.

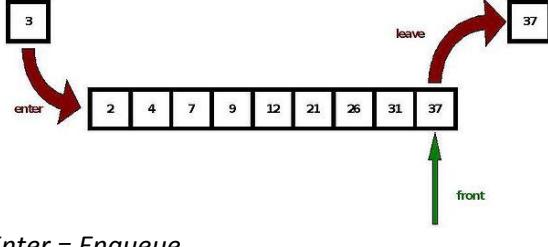
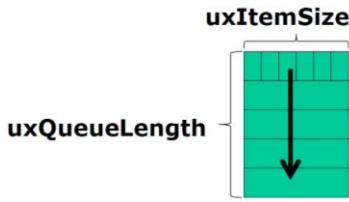
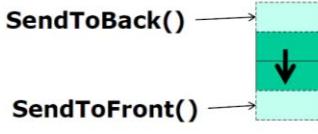
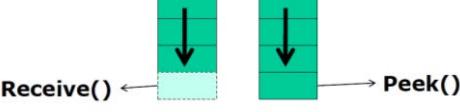
- Semaphore
- Mutual Exclusion / Mutex
 - Spezieller Typ einer (binären) Semaphore
 - Nur ein einziger Task befindet sich gleichzeitig in der Critical Section

?S = Task will Semaphore sperren (Lock), LS = Lock Semaphore; US = Unlock Semaphore

Problem: Priority Inversion				
<p>Task mit hoher Priorität wird von Task mit niedriger Priorität geblockt</p>				
<p><u>Annahmen für Auftreten</u></p> <ul style="list-style-type: none"> • Fixed-priority preemptive scheduling • Single Core Prozessor • Binäre Semaphore (Mutex) 				
<p><u>Typen von Blocking</u></p> <ul style="list-style-type: none"> • Direct <ul style="list-style-type: none"> ◦ Stellt die Konsistenz von gemeinsamen Daten sicher • Push-through <ul style="list-style-type: none"> ◦ Verhindert unendliches blocken durch Priority Inversion • Transitive <ul style="list-style-type: none"> ◦ Beispiel: Job J1 wird von Job J2 geblockt, welcher seinerseits von Job J3 geblockt wird. 				
<p>Priority Inheritance (Prioritätsvererbung, S. 10-12 SW07c Folien)</p> <ul style="list-style-type: none"> • Eine mögliche Lösung für das Priority Inversion Problem! • Idee: Task mit niedriger Priorität erbt die Priorität von jedem Task mit höherer Priorität, welcher auf die Semaphore wartet. Sobald Samphore wieder freigegeben wird, wechselt die Priorität wieder auf die niedrige Priorität. • Deadlocks aber immer noch möglich! 				
Priority Ceiling (Lösung für Priority Inversion Problem)				
<ul style="list-style-type: none"> • Verhindert Deadlocks • Ressource (Daten etc. → Semaphore) ist mit Priorität verknüpft • Die Priority Ceiling wird durch Task mit der höchsten Priorität definiert, welcher die Ressource benötigt • Priorität ist Taskverarbeitungs Ressource zugewiesen 				
<p>Priority Ceiling Regeln und Beispiele siehe SW07c Folien S. 16-18</p>				
FreeRTOS Semaphores: Arten				
<table border="0"> <tbody> <tr> <td style="vertical-align: top;"> <ul style="list-style-type: none"> • Binäre Semaphore • Counting Semaphore • Mutex • Rekursive Mutex </td><td style="vertical-align: top;"> <ul style="list-style-type: none"> vSemaphoreCreateBinary() vSemaphoreCreateCounting() vSemaphoreCreateMutex() vSemaphoreCreateRecursiveMutex() </td><td style="vertical-align: top;"> <ul style="list-style-type: none"> einzelnes Symbol mehrere Symbole Spezielle binäre Semaphore; löst Priority Inversion Problem Kann mehrere Male vom selben Task gelockt werden </td></tr> </tbody> </table>		<ul style="list-style-type: none"> • Binäre Semaphore • Counting Semaphore • Mutex • Rekursive Mutex 	<ul style="list-style-type: none"> vSemaphoreCreateBinary() vSemaphoreCreateCounting() vSemaphoreCreateMutex() vSemaphoreCreateRecursiveMutex() 	<ul style="list-style-type: none"> einzelnes Symbol mehrere Symbole Spezielle binäre Semaphore; löst Priority Inversion Problem Kann mehrere Male vom selben Task gelockt werden
<ul style="list-style-type: none"> • Binäre Semaphore • Counting Semaphore • Mutex • Rekursive Mutex 	<ul style="list-style-type: none"> vSemaphoreCreateBinary() vSemaphoreCreateCounting() vSemaphoreCreateMutex() vSemaphoreCreateRecursiveMutex() 	<ul style="list-style-type: none"> einzelnes Symbol mehrere Symbole Spezielle binäre Semaphore; löst Priority Inversion Problem Kann mehrere Male vom selben Task gelockt werden 		

<p><i>Binäre und Counting Semaphore</i></p> <ul style="list-style-type: none"> „Queue ohne Daten“ Für Mutex verwendet: Immer mit Return Für Synchronisation verwendet: Normalerweise verworfen 	<p><i>Normale und rekursive Mutex</i></p> <ul style="list-style-type: none"> „Queue ohne Daten“ Semaphore mit Priority Inheritance
FreeRTOS Semaphores: Creation	
<p><u>Beispiel Code</u></p> <pre>xSemaphoreHandle sem = NULL; (void)pvParameters; /* parameter not used */ FRTOS1_vSemaphoreCreateBinary(sem); if (sem==NULL) { /* semaphore creation failed */ for(;;){} /* error */ }</pre>	<p><i>Semaphoren sind Queues!</i></p> <pre>#define vSemaphoreCreateBinary(xSemaphore) { \ xSemaphore = xQueueCreate(\ (unsigned portBASE_TYPE)1, \ semSEMAPHORE_QUEUE_ITEM_LENGTH); \ if(xSemaphore != NULL)\ (void)xSemaphoreGive(xSemaphore); \ } \ }</pre> 
FreeRTOS Semaphores: Give & Take	
<p><u>Give (Semaphore freigeben)</u></p> <pre>#define xSemaphoreGive(xSemaphore) xQueueGenericSend((xQueueHandle)(xSemaphore), NULL, semGIVE_BLOCK_TIME, queueSEND_TO_BACK)</pre>	<p><u>Take (Semaphore sperren)</u></p> <pre>#define xSemaphoreTake(xSemaphore, xBlockTime) xQueueGenericReceive((xQueueHandle)(xSemaphore), NULL, (xBlockTime), pdFALSE)</pre>
Beispiel eines Ablaufs bei Semaphore Nutzung (von oben nach unten, von links nach rechts)	
Beispiel mit Semaphores und Interrupts (per Interrupt gesteuerte Semaphore)	

14.2.5. Queues

Was ist eine Queue?	
<p>Eine Queue ist eigentlich nichts anderes als seine Liste, welche meistens als FIFO konfiguriert wird.</p> <p>Eigenschaften</p> <ul style="list-style-type: none"> • Liste aus Elementen • Feste Elementgrösse/ItemSize (@ Erstellzeit) • Feste Queue Grösse (Queue Länge) • Queued mittels kopieren • Spezielle Routinen für ISR Benutzung 	 <p>Enter = Enqueue Leave = Dequeue</p>
FreeRTOS Queue-Funktionen (FreeRTOS Queue API)	
<p>Queue erstellen</p> <pre>xQueueHandle xQueueCreate(unsigned portBASE_TYPE uxQueueLength, unsigned portBASE_TYPE uxItemSize);</pre> <p>Queue löschen</p> <pre>void vQueueDelete(xQueueHandle xQueue);</pre>	
<p>Daten Queue senden</p> <pre>portBASE_TYPE xQueueSendToBack(...); portBASE_TYPE xQueueSendToFront(...);</pre>	
<p>Daten aus Queue auslesen</p> <pre>portBASE_TYPE xQueueReceive(...); portBASE_TYPE xQueuePeek(...);</pre>	
Eigene Queue implementieren	
<p><u>WICHTIG:</u> C-File nicht <i>Queue.c</i> nennen, da das RTOS bereits ein C-File mit diesem Namen erzeugt!</p> <p>Initialisierung</p> <pre>void QUEUE_Init(void) { queueHandle = FRTOS1_xQueueCreate(QUEUE_LENGTH, QUEUE_ITEM_SIZE); if (queueHandle==NULL) { for(;;){ /* no queue generated, out of memory? */ } } }</pre>	
<p>Datend in Queue speichern</p> <pre>void QUEUE_SendMessage(const char_t *msg) { char_t *ptr; size_t bufSize; bufSize = UTIL1_strlen(msg)+1; ptr = FRTOS1_pvPortMalloc(bufSize); // Speicher alloc. UTIL1_strcpy(ptr, bufSize, msg); if (FRTOS1_xQueueSendToBack(queueHandle, &ptr, portMAX_DELAY)!=pdPASS) { for(;;){ /* couldn't write data into queue, not very good to wait here endless but prevents "heuhaufenüberflieger" */ } }</pre>	<p>Daten aus Queue auslesen</p> <pre>const char_t *QUEUE_ReceiveMessage(void) { const char_t *ptr; portBASE_TYPE res; res = FRTOS1_xQueueReceive(queueHandle, &ptr, 0); if (res==errQUEUE_EMPTY) { return NULL; } else { return ptr; } }</pre>

14.2.6. FreeRTOS Trace

Mit FreeRTOS Trace kann die Ausführung der Tasks sowie das locken/unlocken von Semaphoren/Mutex beobachtet werden. Dies kann dabei helfen, Fehler in der Software zu finden oder die Software zu optimieren.

Runtime Statistiken

- Kein richtiges Trace Utility
- Performance Timer hat höhere Auflösung als Tick Timer (bspw. Faktor 10 → 10ms vs. 1ms)
- Misst die Task Ausführungszeit
- Wird mit dem Shell „status“ Befehl zusätzlich zu anderen Informationen ausgegeben

FreeRTOS Hooks (an diesen Punkten wird ein Trace-Operation ausgelöst)

- Makros an strategischen Orten im FreeRTOS
 - z.B. bei Task create, delete, queue receive failed, ...
- User kann bestimmen, wie sich das tracen verhält
 - LED toggeln
 - Informationen nach draussen streamen (SCI, I²C, ...)
 - In RAM Buffer tracen (Percepio FreeRTOS+ Trace)
- FreeRTOS Trace implementiert FreeRTOS Hooks

Wie funktioniert es?	
<ul style="list-style-type: none">• Verwendet RTOS Hooks Makros• Informationen im RAM speichern• RAM an Host (z.B. PC) ausgeben und mit offline viewer anzeigen	<p>Das Diagramm zeigt den Prozess des FreeRTOS Tracing. Auf dem Target System (links dargestellt) befindet sich die Embedded Application, die auf FreeRTOS basiert. Ein Recorder ist Teil des FreeRTOS. Der Debugger auf dem Target System überträgt User Events und FreeRTOS Events in einen RAM buffer. Der RAM buffer hat eine Größe von 4 bytes/event und nimmt etwa 1-2 % des CPU-Budgets ein. Von dort aus wird der RAM dump file erstellt, welcher schließlich in den FreeRTOS+Trace übertragen wird. Der Host PC (rechts dargestellt) enthält einen Debugger, der den RAM dump file überträgt, um es in einem offline viewer anzeigen zu können.</p>
Buffer Size	
<p>Die Buffergrösse (Event Buffer Size) muss je nach RAM Grösse im Processor Expert angepasst werden:</p> <ul style="list-style-type: none">• HCS08 maximal 2kB für Buffer (nur 4kB RAM vorhanden)	

15. Motor

15.1. Motor Signals

PWM: Drehzahl	
	<ul style="list-style-type: none"> • Low-active • PWM Output Compare • 0...100% Duty Cycle → Drehzahl <ul style="list-style-type: none"> ○ Spannung am Motor proportional zum Duty Cycle; Drehzahl proportional zur Spannung • PWM Auflösung <ul style="list-style-type: none"> ○ Muss kleiner als die kleinste Zeitkonstante des Prozesses sein ○ Kleiner als mechanische Zeitkonstante des Motors (Zeit, bei welcher Motor 63% Speed @ U_{Nominal} erreicht)
DIR: Drehrichtung	
Normales GPIO, mit welchem die Drehrichtung angegeben wird (z.B. 0 = vorwärts, 1 = rückwärts)	
C1 & C2: Quadrature Encoder Signals	
	Messsignal des Quadratur Encoders (mehr dazu siehe Kapitel 15.3)
Beispiel Implementation	
	<pre> typedef enum { MOT_DIR_FORWARD, /*!< Motor forward */ MOT_DIR_BACKWARD /*!< Motor backward */ } MOT_Direction; void MOT_SetDirection(MOT_Direction dir) { if (dir == MOT_DIR_FORWARD) { DIR_ClrVal(); } else { DIR_SetVal(); } } MOT_Direction MOT_GetDirection(void) { if (DIR_GetVal() == 0) { return MOT_DIR_FORWARD; } else { return MOT_DIR_BACKWARD; } } </pre>
<u>PWM (Interfaces):</u> <pre> static void MOT_SetDutyRatioPercent (MOT_SpeedPercent percent) {} static void MOT_SetSpeedPercent (MOT_SpeedPercent percent) {} void MOT_ChangeSpeedPercent (MOT_SpeedPercent relPercent) {} </pre>	

strcmp(), sizeof() & atoi() ???

(S. 28-29 Motor Signals Folien)

15.2. Motor Trace

Ziel: Drehrichtung, PWM Duty-Cycle und weiteres über Shell ausgeben.

Beispiel Implementation

```
static portTASK_FUNCTION(TraceTask, pvParameters) {}      // Trace Task definieren

static void PID_PrintHelp(const FSSH1_StdIOType *io) {
    FSSH1_SendHelpStr("trace", "Group of trace commands\r\n", io->stdOut);
    FSSH1_SendHelpStr(" help|status", "Shows trace help or status\r\n", io->stdOut);
#ifndef PL_HAS_I2C
    FSSH1_SendHelpStr(" none|shell|i2c", "Sets the trace channel to be used\r\n", io->stdOut);
#endif // und weitere #if-#endif einfügbar
}

static void PID_PrintStatus(const FSSH1_StdIOType *io) {
    FSSH1_SendStatusStr("Trace", "\r\n", io->stdOut);
    FSSH1_SendStatusStr(" channel", "", io->stdOut);
    if (traceChannel==TRACE_TO_NONE) {
        FSSH1_SendStr("NONE\r\n", io->stdOut);
    } else if (traceChannel==TRACE_TO_SHELL) {
        FSSH1_SendStr("SHELL\r\n", io->stdOut);
    }
#ifndef PL_HAS_MOTOR_SIGNALS
    if (traceMotor) {
        FSSH1_SendStr("motor(on) ", io->stdOut);
    } else {
        FSSH1_SendStr("motor(off) ", io->stdOut);
    }
#endif // und weitere #if-#endif einfügbar
}

uint8_t TRACE_ParseCommand(const char *cmd, bool *handled, const FSSH1_StdIOType *io) {}
```

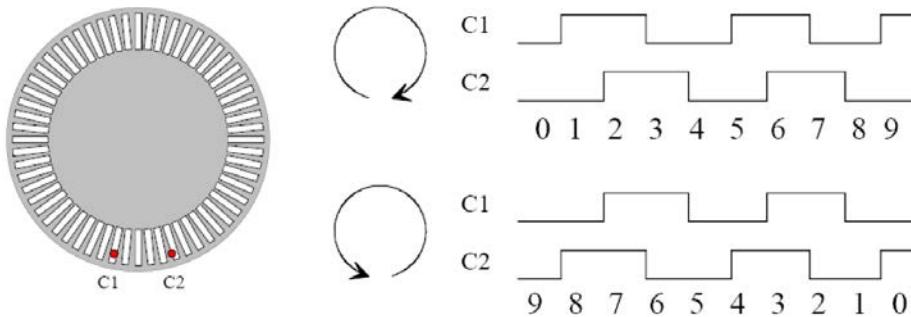
Beispiel einer Ausgabe

```
trace          ; Group of trace commands
help|status    ; Shows trace help or status
none|shell     ; Sets the trace channel to be used
motor on|off   ; Enables or disables motor trace
```

```
CMD> trace status
Trace :
    channel : NONE
    options : motor(on)
CMD> trace shell
    => Motor fw 0x0000;
    => Motor fw 0x0000;
SW1 pressed!
    => Motor fw 0x0CCB;
SW1 released!
    => Motor fw 0x0CCB;
    => Motor fw 0x0CCB;
```

15.3. Quadrature Encoder

Der Quadratur Encoder besteht aus einer rotierenden Scheibe und zwei optischen Sensoren, welche versetzt zueinander angeordnet sind (rote Punkte). Durch das versetzte Anordnen wird nur eine Scheibe für zwei Signale benötigt anstatt zwei Scheiben.



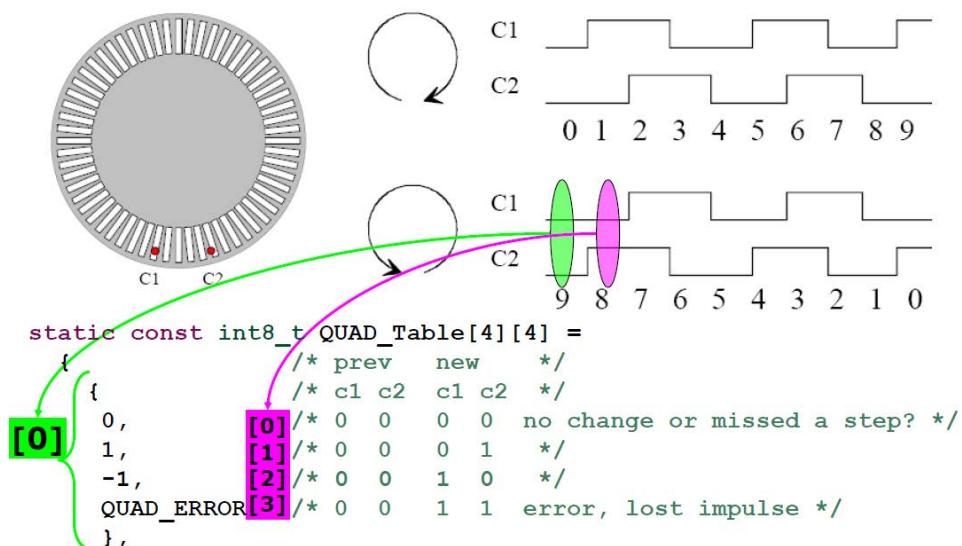
Achtung bei Real Time: Die maximale Signalrate (bei maximaler Drehzahl) muss berücksichtigt werden). Diese Signalrate wird folgenderweise bestimmt: Maximale Drehzahl*Anzahl Zustände pro Umdrehung.

Möglichkeiten das Signal abzutasten

- Interrupts
 - Interrupt Latenz beachten
 - System load beachten
- Sampling
 - Samples/s im Minimum 2x Signalrate (Nyquist-Shannon Theorem)
 - Beim INTRO ergeben sich 4*100 Signale pro Umdrehung des Motors

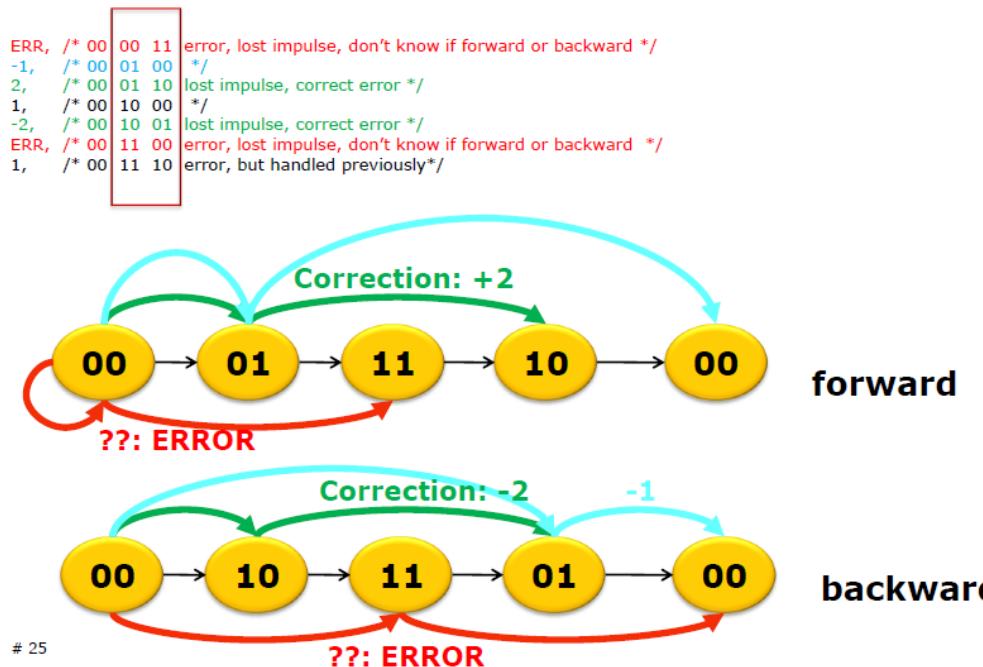
15.4. Quadrature Decoder

Die abgetasteten Signale werden mit dem vorherigen abgetasteten Wert verglichen (siehe Kommentare unten). Somit kann die Drehrichtung bestimmt werden. Falls ein Schritt zu gross sein sollte, kann zusätzlich eine Error Correction durchgeführt werden.



Error Correction

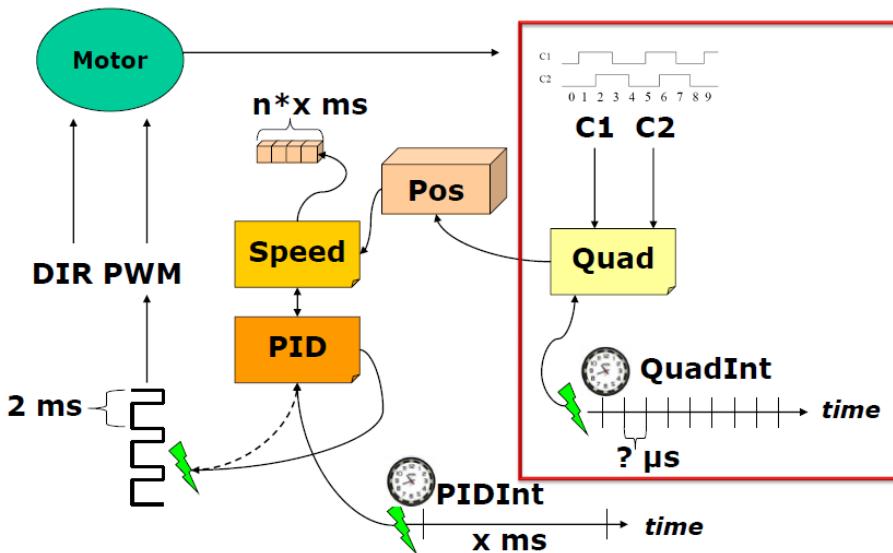
Die Fehler-Korrektur basiert darauf, dass normalerweise kein Zustandsübergang ausgelassen werden darf. Dazu werden drei Zustände (jetzt, vorherig und vor-vorherig) gespeichert und verglichen. Dabei ergibt sich folgendes Muster für die Fehlerauswertung:



Es kann unterschieden werden, ob ein Error vorliegt oder nur ein Impuls verloren gegangen ist (grüne Pfeile).

System Architektur und Timing (Encoder/Decoder = Roter Bereich)

Die Positionswerte des Quadratur Encoders werden in einem Array gespeichert. Diese n-Werte werden über eine $n \times x$ -Zeit gemittelt (Tacho), um eine verlässlichere Geschwindigkeitsangabe zu erhalten. Dieser Wert wird schliesslich an den PID-Regler weitergeleitet, welcher das PWM-Signal generiert (Closed Loop Control).



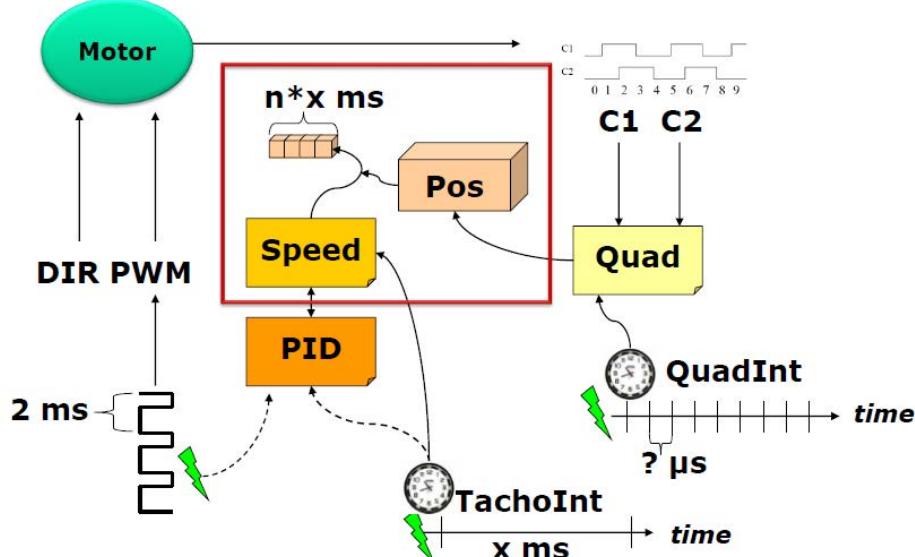
Real-time Aspekte des Quadratur Decoders

- System Load kann zunehmen durch..
 - Timer
 - PWM-Timer
 - Quadrature Sampling
 - Interrupts...
 - So schnell wie möglich
 - So kurz wie möglich
 - Komplexe Berechnungen/Funktionsaufrufe innerhalb der ISR vermeiden
 - Benutzung von.. (bei Problemen oder um das Timing zu verifizieren)
 - Oszilloskop
 - Logic Analyzer

15.5. Tacho

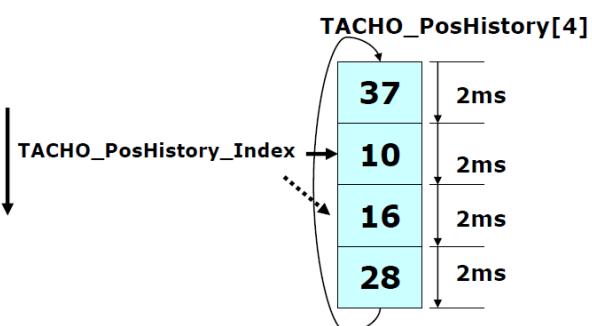
System Architektur und Timing (Tacho = Roter Bereich)

Die n-Positionswerte werden über eine $n \cdot x$ -Zeit gemittelt; um eine verlässlichere Geschwindigkeitsangabe zu erhalten.



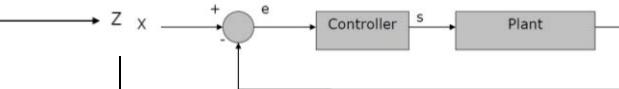
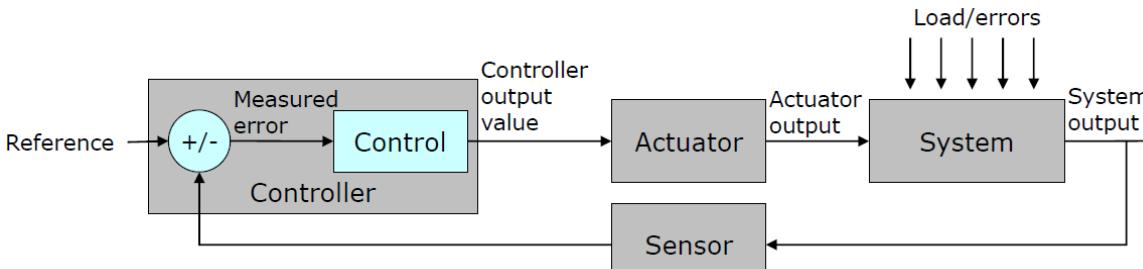
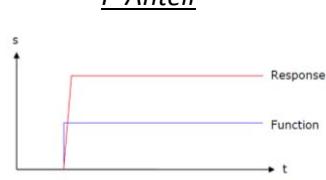
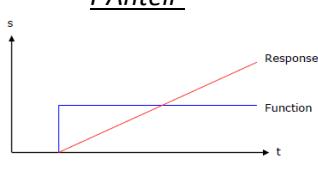
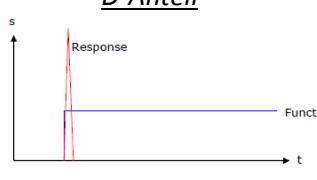
TACHO Sample() position sampling

TACHO_Sample speichert die aktuelle Position des Motors (anhand QuadraturEncoderWert) in einem Ringbuffer. Dies geschieht bei jedem Tacho-Interrupt (hier alle 2ms). Der TACHO_PosHistory_Index zeigt auf das nächste freie Element (ist aber kein Pointer sondern



ein „static volatile uint8_t“!).	
TACHO_CalcSpeed()	
Berechnet die Geschwindigkeit des Motors. Diese Funktion möglichst „on demand“ aufrufen, also nur wenn man die Geschwindigkeit jetzt benötigt. Grund: Die Berechnung ist aufgrund dem Mitteln der Werte über z.B. 10 Werte relativ zeitaufwändig.	<ul style="list-style-type: none"> - Known time (delta) distance between <ul style="list-style-type: none"> - Current position - Position to be overwritten - Steps/second calculation - # steps per second
<pre>delta = old-new; /* delta of oldest position and most recent one */ /* calculate speed. this is based on the delta and the time (number of samples or entries in the history table) */ speed = (int32_t)(delta * 1000/(SPEED_CALC_PERIOD_MS*(NOF_HISTORY-1)));</pre>	
Schritt Arithmetik: Overflow/zero	
<p>Beim Berechnen der Geschwindigkeitswerte muss bei einem Overflow aufgepasst werden, dass nicht falsche Werte berechnet werden.</p> <p><u>Beispiele (4 Bit Signed):</u></p> <p>curr -8 = , prev = 0 $0 - (-8) = 8 = 0b1000 = -8$</p> <p>curr 7 = , prev = -7 $-7 - 7 = 14$ (anstatt -2)</p>	<ul style="list-style-type: none"> - delta = curr - prev - signed arithmetic!

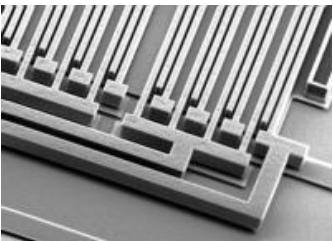
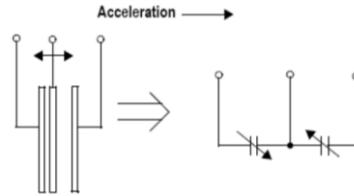
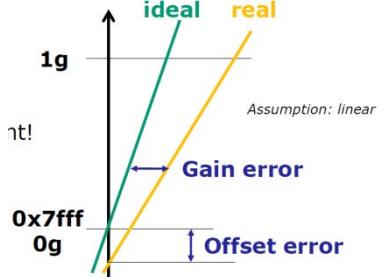
15.6. Closed Loop Control (PID Regulator)

Control vs. Closed Loop Control		
Control (Steuerung)  <ul style="list-style-type: none"> • Übertragung von Informationen • Bearbeitung von Informationen • Sensoren als Informationsproduzenten • Aktuatoren als Informationskonsumenten <p><u>Eigenschaften:</u></p> <ul style="list-style-type: none"> • Open Loop (offener Regelkreis) • Kein Feedback 	Closed Loop Control (Regelung)  <ul style="list-style-type: none"> • Geschlossener Regelkreis (closed loop) • Formelles Modell für <ul style="list-style-type: none"> ○ Messen – Vergleichen – Steuern • X = Sollwert (Input) • Z = Istwert / Messwert (Output) • e = Abweichung (X-Z), Regelfehler • Prozesskontroll Algorithmus ist im „Controller“ enthalten • Der Controller stellt einen neuen Stellwert „s“ für das System (Plant) 	
Systemarchitektur		
 <p>Actuator output = PWM Signal System = Motor Sensor = Quadratur Encoder + Tacho <u>Ziel:</u> Definiertes Verhalten des Systems, Berücksichtigung von Störeinflüssen</p>		
Reglerauslegung		
<p>Normalerweise wird der Regler anhand der Sprungantwort des System Outputs ausgelegt.</p> <p><u>Regler-Anteile</u></p> <p>P = Proportional-Anteil → Einfache Verstärkung des Regelfehlers I = Integral-Anteil → Summiert den Regelfehler auf, eliminiert stationäre Ungenauigkeit D = Differential-Anteil → Steigung des Regelfehlers, schnelle Änderungen werden hoch verstärkt</p>		
<p><u>P-Anteil</u></p>  $s(t) = K_p \cdot e(t)$	<p><u>I-Anteil</u></p>  $s(t) = K_i \int_0^t e(t') \cdot dt'$	<p><u>D-Anteil</u></p>  $s(t) = K_d \frac{d}{dt} e(t)$

Reglerauslegung: PID-Regler	
<p><u>Beispiel-Code</u></p> <pre>esum = esum+e; s = Kp*e + Ki*Ta*esum + Kd*(e-eprev)/Ta; eprev = e;</pre> <p><u>Wofür welche Anteile?</u></p> <p>P+D Anteil: Geschwindigkeit des Reglers I: Stationäre Genauigkeit</p> <p><u>Sonstiges</u></p> <ul style="list-style-type: none"> • I-Anteil benötigt Anti-Wind-Up, damit der I-Anteil mit der Zeit nicht zu gross wird und die Systemantwort nicht ins unendliche wächst. • Totzeit möglichst klein halten! (durch angemessenes Timing z.B. jede PWM-Periode einen neuen Stellwert generieren) 	<p><u>Sprungantwort</u></p> $s(t) = K_p \cdot e(t) + K_d \frac{d}{dt} e(t) + K_i \int_0^t e(t') \cdot dt'$

16. Accelerometer

Ein 3-Axis-Accelerometer (3-Achsen-Beschleunigungssensor) generiert anhand der Beschleunigung ein Ausgangssignal für jede Achse (X-Y-Z).

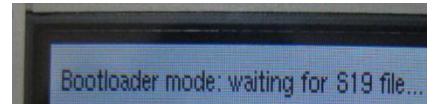
Aufbau von Beschleunigungssensoren	
 	<p>Accelerometer sind meistens sogenannte MEMS (Micro-Electro-Mechanical System). Durch eine von aussen wirkende Beschleunigung entsteht ein Unterschied in den Kapazitäten zwischen den einzelnen Finnen. Diese Kapazität kann gemessen werden und somit auch die Beschleunigung.</p>
<p>Weiteres zur Hardware siehe „Hardware stuff“</p>	
Eigenschaften	
<p><u>Typische Interfaces</u></p> <ul style="list-style-type: none"> • Analog (AD) • Digital (SPI, I2C) <p><u>Kalibration pro Kanal benötigt (X-Y-Z) wegen:</u></p> <ul style="list-style-type: none"> • Gain-Fehler • Offset-Fehler • Wegen Temperatur- und Herstellungseinflüssen • Kalibrationsdaten können im Processor Expert eingegeben werden (falls keine Kalibration durchgeführt wird → „Default Werte“) und werden dann zu Kompilierzeit ins Flash / ROM geschrieben (INTRO Projekt spezifisch) 	
Beispiel Implementation	
<p><u>Interfaces</u></p> <pre> void ACCEL_GetValues(int16_t *x, int16_t *y, int16_t *z); uint8_t ACCEL_ParseCommand(const char *cmd, bool *handled, const FSSH1_StdIOType *io); void ACCEL_Init(void); </pre> <p><u>Aus Accel.c</u></p> <pre> void ACCEL_GetValues(int16_t *x, int16_t *y, int16_t *z) { *x = ACCEL1_GetXmg(); *y = ACCEL1_GetYmg(); *z = ACCEL1_GetZmg(); } /* Gibt den Status aus wie z.B. die aktuellen Beschleunigungswerte*/ static void ACCEL_PrintStatus(const FSSH1_StdIOType *io) {} /* Gibt Informationen zu den verfügbaren Shell-Funktionen aus */ static void ACCEL_PrintHelp(const FSSH1_StdIOType *io) {} /* Übersetzt die Shell-Befehle (muss in Shell.c unter SHELL_ParseCommand() eingefügt werden)*/ uint8_t ACCEL_ParseCommand(const char *cmd, bool *handled, const FSSH1_StdIOType *io) {} </pre>	

17. LCD

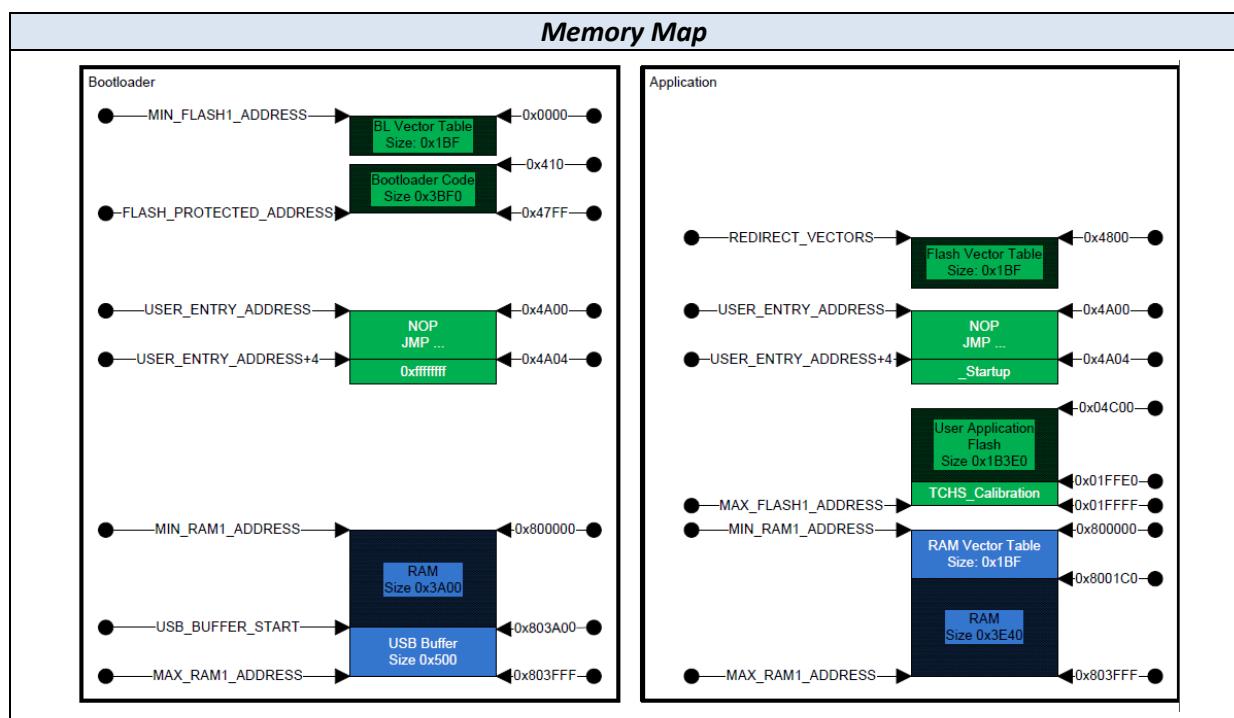
17.1. Bootloader

Mit einem Bootloader kann die Firmware eines Devices (hier Microcontroller der das LCD steuert) auf das Device geladen werden. Folgendes wird für den Bootloader benötigt:

- S19-File (Text File → Code)
- Device muss sich im „Bootloader mode“ befinden



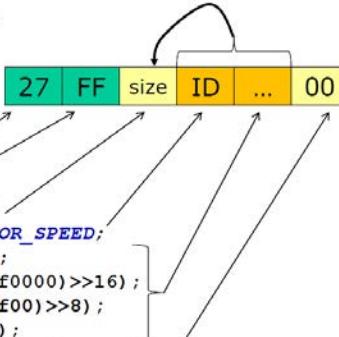
Das Device wird z.B. mittels USB am PC angeschlossen. Dabei erscheint das Device als Wechselmedium im Arbeitsplatz. Nun kann das S19-File mittels Drag&Drop oder kopieren auf das Wechselmedium verschoben werden. Die Aplikation wird jetzt automatisch geflasht. Nun das Board resetten und fertig.



17.2. I²C

Beispiel einer Implementation (Format einer Message)

```
void I2C_SendTachoSpeed(void) {
    int32_t speed;
    char buf[9];
    uint16_t snt;
    speed = TACHO_GetSpeed();
    buf[0] = I2C_PRE_BYTE0;
    buf[1] = I2C_PRE_BYTE1;
    buf[2] = 5; /* data length */
    buf[3] = (uint8_t) I2C_MSG_MOTOR_SPEED;
    buf[4] = (uint8_t) (speed>>24);
    buf[5] = (uint8_t) ((speed&0xffff0000)>>16);
    buf[6] = (uint8_t) ((speed&0xff00)>>8);
    buf[7] = (uint8_t) (speed&0xff);
    buf[8] = 0; /* termination byte */
    I2C2_SelectSlave(I2C_LCD_ADDR);
    I2C2_SendBlock(&buf[0], sizeof(buf), &snt);
}
```



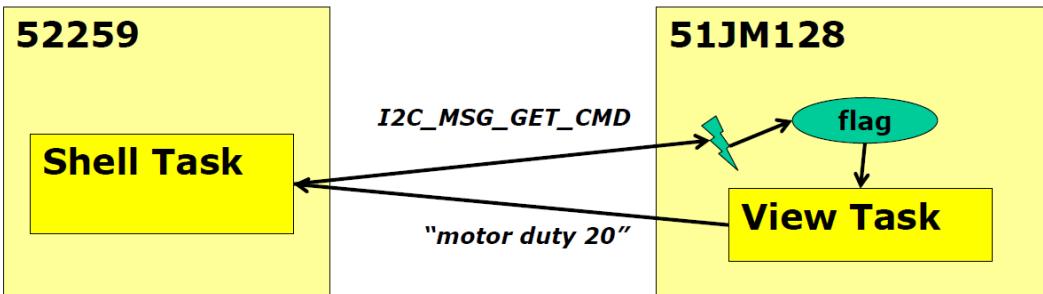
Mit den Messages kann auch auf das LCD Modul getraced werden. Dabei werden dann z.B. die Motorgeschwindigkeit oder die Werte des Beschleunigungssensors auf dem LCD ausgegeben.

17.3. C++ and Slider

Für eine ausführlichere Beschreibung von C++ siehe „SW10b C++ Slider“ slides.

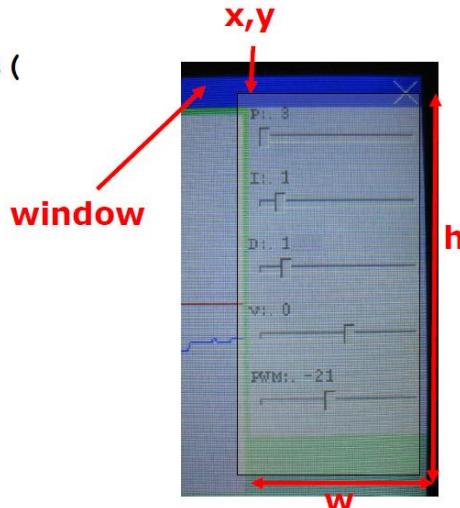
I2C Data Flow

- 52259 (Tower) fragt 51JM128 (LCD) nach Befehlen/Optionen (im Bild „motor duty“ = 20)
- Normale Shell commands
 - Limitierte cmd buffer Länge und I²C Paketgrösse
 - JM128 sendet einen Befehl nach dem anderen
- Limitierung
 - Command/Data Flow nur unidirektional



Slider Creation

```
void SLIDER_CreateSliders(
    UI1_Window *window,
    UI1_PixelDim x,
    UI1_PixelDim y,
    UI1_PixelDim w,
    UI1_PixelDim h)
{
    ...
}
```

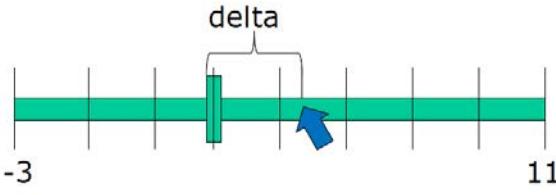


Window Callback

Diese Funktion fragt ab, welcher Event ausgeführt werden muss und führt ihn aus.

```
void SLIDER_SliderW_WindowCallback(
    UI1_Window *window, UI1_Element *element,
    UI1_EventCallbackKind kind,
    UI1_Pvoid data)
{
    if (kind==UI1_EVENT_CLICK) {
        sliderP.OnClick((UI1_Coordinate*)data);
        /* other objects? */
    } else if (kind==UI1_EVENT_CLICK_MOVE) {
        sliderP.OnClickMove((UI1_Coordinate*)data);
    } else if (kind==UI1_EVENT_PAINT) {
        sliderP.Paint();
    }
}
```

Command Getter	
Holt den entsprechenden Befehl vom LCD-Modul, welcher über einen Slider eingegeben wurde.	<pre>void SLIDER_GetCmdString(unsigned char *buf, size_t bufSize) { SliderValT val; static SliderValT oldKp=-1, oldKd=-1; for(;;) { // breaks val = sliderP.GetValue(); if (val!=oldKp) { UTIL1_strcpy(buf, bufSize, (const unsigned char*)"pid kp "); UTIL1_strcatNum32s(buf, bufSize, val); oldKp = val; break; } buf[0] = '\0'; // empty response break; } }</pre>
Paint()	
Stellt die Slider mit dem Balken, den Knöpfen (Knobs) sowie Stellung des Knobs auf dem LCD dar.	<pre>void SliderWidget::Paint(void) { // background UI1_DrawFilledBox(m_window, m_area.x, m_area.y, m_area.w, m_area.h, UI1_COLOR_BRIGHT_BLUE); // slider horizontal line UI1_DrawHLine(m_window, m_area.x+SLIDER_H_BORDER, m_area.y+(m_area.h/2)-2, m_area.w-(SLIDER_H_BORDER*2), UI1_COLOR_BRIGHT_GREY); ... // draw slider knob knobArea = GetKnobPosition(); UI1_DrawFilledBox(m_window, knobArea.x+1, knobArea.y+1, knobArea.w-2, knobArea.h-2, UI1_COLOR_BRIGHT_BLUE); ... // write label and value buf[0] = '\0'; UTIL1_strcpy(buf, sizeof(buf), m_label); ... x = (UI1_PixelDim)(m_window->prop.x + m_area.x); y = (UI1_PixelDim)(m_window->prop.y + m_area.y); FDisp1_WriteString(buf, UI1_COLOR_BLACK, &x, &y, Cour08n_GetFont()); }</pre> <p># 14</p>
GetKnobPosition()	
Liest die aktuelle Position des Knopfes aus und gibt sie zurück.	<pre>WidgetArea SliderWidget::GetKnobPosition(void) { WidgetArea area; int i; area.w = KNOB_WIDTH; area.h = m_area.h-(KNOB_H_BORDER*2); if (area.h>KNOB_HEIGHT) { area.h = KNOB_HEIGHT; // max height } i = (int)((GetVal()-GetMin())/((GetMax()-GetMin())/m_steps)); // i is in range 0..m_steps area.x = SLIDER_H_BORDER+(UI1_PixelDim)(m_area.x+(m_area.w*i/m_steps)); area.x -= area.w/2; // center know at pixel position if (area.x<m_area.x+SLIDER_H_BORDER) { // completely to the left: start at left side area.x = m_area.x+SLIDER_H_BORDER; } else if (area.x>m_area.x+m_area.w-SLIDER_H_BORDER-area.w) { // completely to the right: adjust position area.x = m_area.x+m_area.w-SLIDER_H_BORDER-area.w; } area.y = m_area.y + (m_area.h/2) - area.h/2; // center return area; }</pre>

<i>SetKnobPosition()</i>	
<p>Setzt die Position des Knopfes. Wird bei der Initialisierung sowie beim Klicken auf das LCD benutzt.</p>	<pre>void SliderWidget::SetKnobPosition(WidgetArea *clickPos) { SliderValT delta; WidgetArea currKnob; currKnob = GetKnobPosition(); delta = clickPos->x - currKnob.x; // get difference in pixels delta *= (GetMax()-GetMin())/m_area.w; // scale to value SetVal(GetVal()+delta); }</pre>  <p>The diagram shows a horizontal slider with a green track and a blue knob. The track has tick marks at integer intervals from -3 to 11. The current position of the knob is at the tick mark for 5. A blue arrow points to the gap between the current position and the previous tick mark at 4. A bracket above the arrow is labeled 'delta', indicating the change in position.</p>

18. Radio Tranceiver

SPI Interface	
<p>MISO = Master In – Slave out MOSI = Master Out – Slave In CE = Chip Enable CLK = Clock IRQ = Interrupt Request (Wakeup Microcontroller) RESET = Reset Tranceiver ATTN = Wakeup transceiver RTTX = Antenna switch</p>	
SPI Protocol	
<p>Die SPI Schnittselle kann auf grundsätzlich vier verschiedene Arten konfiguriert werden. Die Abbildung rechts zeigt was die Einstellungen der „Clock edge“ und „Shift clock idle polarity“ bewirken. Im INTRO wird die rot umrahmte Version benutzt und ist im Allgemeinen die am häufigsten genutzte.</p>	
Mögliche Device Topologie	
Processor Expert Einstellungen	
<ul style="list-style-type: none"> • Buffer Size nach eigenem Ermessen einstellen. Da keine allzu grossen Strings verschickt werden müssen, kann sie ggü. dem Standardwert stark reduziert werden (z.B. auf 20). Dies spart Speicherplatz auf dem Stack. • Interrupt Prioritäten kontrollieren! Sie darf nicht mit einem anderen Interrupt überlappen. • IRQ Pull-Up enablen 	

Radio: Packet handling (Receive Packet)

Execution Flow

```

sequenceDiagram
    participant Transceiver
    participant RadioIRQ [Radio IRQ]
    participant msgQ
    participant Event
    participant mainT
    participant RadioHandleEvent
    participant HandleMsg

    RadioIRQ->>Transceiver: IRQ
    activate Transceiver
    Transceiver->>msgQ: Data Indication Packet Event
    activate msgQ
    msgQ->>Event: e
    activate Event
    Event->>mainT: 
    activate mainT
    mainT->>HandleMsg: HandleMsg()
    activate HandleMsg
    HandleMsg->>RadioHandleEvent: 
    activate RadioHandleEvent
    RadioHandleEvent->>Event: e
    deactivate Event
    deactivate mainT
    deactivate HandleMsg
    deactivate RadioHandleEvent
    
```

1. Inspect Packet
2. Queue message
3. Generate Event

Im MainTask (mainT) wird HandleEvent(e) und RADIO_Handle() ausgeführt.
→ Queue Message über IRQ!
→ Events um die Radio State Machine zu verbessern

RADIO_DataIndicationPacket()

```

void RADIO_DataIndicationPacket(tRxPacket *sRxPacket) {
    if (sRxPacket->u8Status==SMAC1_TIMEOUT) { /* keine neuen Daten? */
        EVNT_SetEvent(EVNT_RADIO_TIMEOUT);
    } else if (sRxPacket->u8Status == SMAC1_SUCCESS) {
        if (RADIO_isSniffing) { /* Funkübertragung sniffen? */
            QueueMessage(RADIO_QUEUE_MSG_SNIFF,
                sRxPacket->pu8Data, sRxPacket->u8DataLength);
        }
        if (RADIO_AppStatus==RADIO_WAITING_FOR_ACK && isACK()) {
            EVNT_SetEvent(EVNT_RADIO_ACK);
        } else if (isMessageForMe()) { /* queue it? */
            EVNT_SetEvent(EVNT_RADIO_DATA);
        } else { /* unknown packet? */
            EVNT_SetEvent(EVNT_RADIO_UNKNOWN);
        }
    } else if (sRxPacket->u8Status==SMAC1_OVERFLOW) {
        EVNT_SetEvent(EVNT_RADIO_OVERFLOW);
    }
}

```

QueueMessage()

Relevanter Code:

```

uint8_t i, buf[RADIO_QUEUE_ITEM_SIZE];
buf[0] = kind; /* kind */
buf[1] = msgSize; /* size */
i = 2;
while(msgSize>0 && i<sizeof(buf)) { /* data */
    buf[i++] = *msg;
    msg++;
    msgSize--;
}

```

kind
size
Data...

RADIO_Handle()	
RADIO_Handle() sendet Daten und bearbeitet die empfangenen Daten (Zugriff auf das Funkmodul über über RADIO_HandleState()). RADIO_HandleMessage() liest die Daten aus dem Radio Buffer aus und übergibt die Daten der Shell (alles nur wenn Sniffing aktiv!)	<pre>void RADIO_Handle(void) { uint8_t buf[RADIO_QUEUE_ITEM_SIZE]; if (RADIO_isOn) { RADIO_HandleState(); } /* poll radio message queue */ if (FRTOS1_xQueueReceive(RADIO_MsgQueue, buf, 0)==pdPASS) { /* received message from queue */ RADIO_HandleMessage(buf); } }</pre>
RADIO_HandleState() State Diagram	
Rechts ist die State-Machine des Radio Tranceivers abgebildet.	

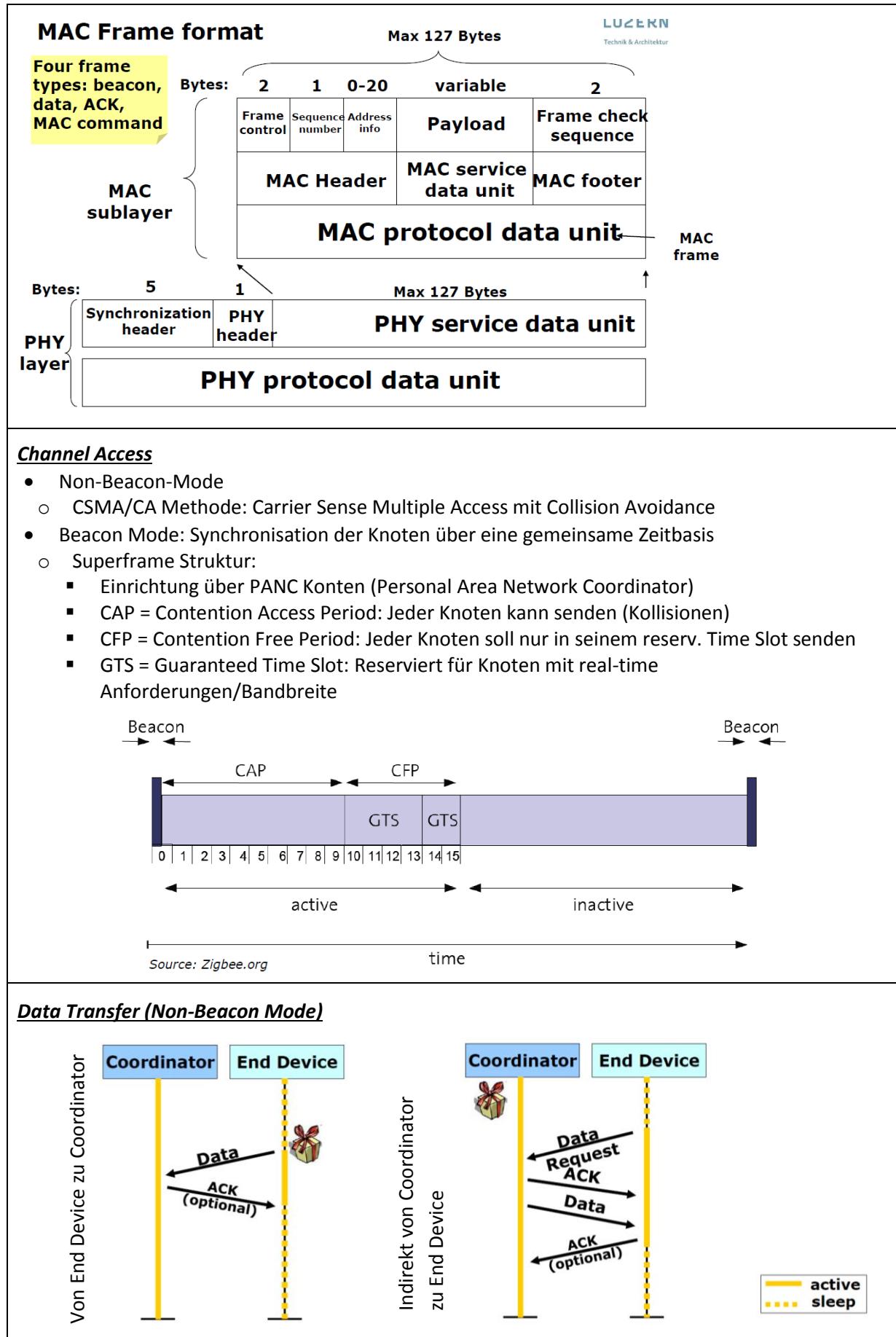
18.1. Remote Control

Mögliche Probleme, Störung der Übertragung													
Wie merke ich, dass die Daten meine eigenen sind? <ul style="list-style-type: none"> • Gleicher Kanal? → Anderen Kanal wählen • Oder eine Art Protokoll einführen. Mit einer individuellen PANID (Previous Access Network Identifier) kann. Zusätzlich kann das Protokoll z.B. mit einer CRC-Summe ergänzt werden. 	<table border="1"> <tr> <td align="center" colspan="4">message</td> </tr> <tr> <td align="center" colspan="2">PANID</td> <td align="center" colspan="2">payload</td> </tr> <tr> <td align="center" colspan="1">PANID</td> <td align="center" colspan="1">kind</td> <td align="center" colspan="1">data</td> <td align="center" colspan="1">CRC</td> </tr> </table>	message				PANID		payload		PANID	kind	data	CRC
message													
PANID		payload											
PANID	kind	data	CRC										
<table border="1"> <tr> <td align="center" colspan="3">'ABC' 'x' value</td> </tr> <tr> <td align="center" colspan="3">PANID kind payload</td> </tr> </table> 	'ABC' 'x' value			PANID kind payload									
'ABC' 'x' value													
PANID kind payload													

19. Wireless (IEEE 802.15.4)

Was ist IEEE 802.15.4?									
<p>IEEE 802.15.4 ist ein Standard, welcher die Physical (PHY) und Medium Access Control (MAC) Layer beinhaltet.</p> <ul style="list-style-type: none"> • Für Wireless Personal Area network (WPAN) • 3 Frequenzbänder • 3dBm minimale Übertragungsleistung (500µW) • Fokus <ul style="list-style-type: none"> ◦ Low Cost: <5\$ ◦ Low Speed: 250kbit/s ◦ Low Power: Batterie tauglich • Realtime: Garantierter Zeit Slot • Integrierte Sicherheit 	<pre> graph TD App[Application] <--> AL[Application Layer (AL) Application Framework (AF) ZigBee Device Objects (ZDO) Application Support Sublayer (ASP)] AL <--> NWK[Network (NWK) Star / Mesh / Cluster - Tree] NWK <--> MAC[Media Access Control (MAC) Device Types, Channel Access] MAC <--> PHY[Physical Interface (PHY) 868 MHz / 915 MHz / 2.4 GHz] </pre>								
Physical (PHY) Layer									
<p><u>Tasks</u></p> <ul style="list-style-type: none"> • Aktivieren/deaktivieren der Transceiver <ul style="list-style-type: none"> ◦ send, receive, sleep • Receiver Energy Detect (ED) <ul style="list-style-type: none"> ◦ Ermittelt Signalstärke • Link Quality Indication (LQI) <ul style="list-style-type: none"> ◦ Qualität des empfangenen Signals • Clear Channel Assessment (CCA) <ul style="list-style-type: none"> ◦ Bestimmt Medium Aktivität: busy oder idle • Auswahl der Kanalfrequenz <ul style="list-style-type: none"> ◦ 27 channels möglich 	<p><u>Frequenzen</u></p> <p>Frequenzen sind für diverse Dienste reserviert (Mobiltelefone, Fernsehen etc.). Für uns interessant: ISM Band ("Industrial, Scientific, Medical"), da lizenfrei</p> <table border="1"> <thead> <tr> <th>Frequency</th><th>Comment</th></tr> </thead> <tbody> <tr> <td>433 – 464 MHz</td><td>Europe</td></tr> <tr> <td>900 – 928 MHz</td><td>Americas</td></tr> <tr> <td>2.4 – 2.5 GHz</td><td>WLAN/WPAN, worldwide</td></tr> </tbody> </table>	Frequency	Comment	433 – 464 MHz	Europe	900 – 928 MHz	Americas	2.4 – 2.5 GHz	WLAN/WPAN, worldwide
Frequency	Comment								
433 – 464 MHz	Europe								
900 – 928 MHz	Americas								
2.4 – 2.5 GHz	WLAN/WPAN, worldwide								
<u>Problem:</u> Knoten will senden, jedoch ist bereits ein anderer Knoten am senden									
<p><u>Lösung:</u></p> <ul style="list-style-type: none"> • CSMA (Carrier Sense Multiple Access) → Wartet, falls jemand schon am Senden ist und versucht es nach einer zufälligen Zeit wieder • CD (Collision Detection) → „Hört zu“, ob eine Kollision stattfindet. Falls ja: Kollisionssignal senden und später neu versuchen (Beispiel: Ethernet) • CA (Collision Avoidance) → Für Wireless Protokolle, wenn nicht gleichzeitig gesendet und empfangen werden kann. Falls Carrier frei: Broadcast zu den anderen Stationen, dass sie nicht senden dürfen. Variante: RTS senden und CTS empfangen 									
<p><u>2.4GHz PHY Frame Format:</u></p> <p>Max. Übertragungszeit: 4.25ms bei 250 Kbps</p>	<p>Bytes: 5 1</p> <table border="1"> <tr> <td>Preamble + Seq. Nr</td><td>Frame Size</td><td>Max 127 Bytes</td><td>Payload</td></tr> <tr> <td colspan="4" style="text-align: center;">PHY protocol data unit</td></tr> </table>	Preamble + Seq. Nr	Frame Size	Max 127 Bytes	Payload	PHY protocol data unit			
Preamble + Seq. Nr	Frame Size	Max 127 Bytes	Payload						
PHY protocol data unit									

Media Access Control (MAC) Layer		
SMAC <ul style="list-style-type: none"> + Kleine Grösse (\approx5Kbyte) + Einfach (wenige API calls, einfache State Machine) + 2.4GHz Band (lizenzfrei) - Routing? Addressing? - Realtime (keine Garantien) - Keine Verschlüsselung - Channels: Welchen wählen? - Low Power? - Verlorene Pakete (neu senden?) 		IEEE 802.15.4 ZigBee Peer, Star, Mesh, Proprietary Mesh, Star, Cluster Interoperability
INTRO Systemübersicht		
Netzwerkstrukturen <ul style="list-style-type: none"> • Peer-to-peer (P2P) • Star <p>PAN = Public Area Network</p>		
Device Types <ul style="list-style-type: none"> • Reduced Function Device (RFD) <ul style="list-style-type: none"> ◦ Einfach und billig, nur Stern-Topologie → z.B. Netzwerk Ecken, Lichtschalter • Full Function Device (FFD) <ul style="list-style-type: none"> ◦ Volle 802.15.4 Unterstützung, kann auch Abschlusskonten sein → z.B. Netzwerk Knoten, Router • Personal Area Network (PAN) Coordinator <ul style="list-style-type: none"> ◦ Spezialfall eines FFD, kennt das gesamte Netzwerk, erstellt Netzwerk ID und Adress-Blöcke 		
Coordinator <p>Jedes Device muss eine Verbindung zu einem anderen Device besitzen. Device mit mehreren Verbindungen = Coordinator. Coordinator: Stellt Synchronisation Services (Beacon/Non-Beacon), Routing etc. zur Verfügung</p>		



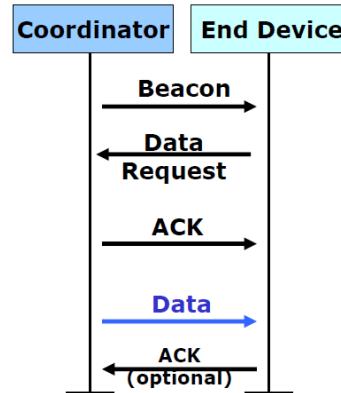
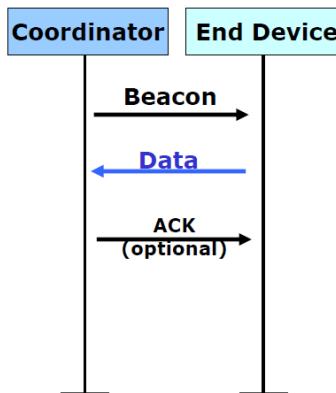
Data Transfer (Beacon Mode)

End Device (ED) zu Coordinator (CORD)

- Beacon wird periodisch gesendet
- Koordinator könnte immer aktiv sein
- End Device kann schlafen (sleep)
- Geringster Energieverbrauch
- Benötigt präzises Timing
- Beacon-Periode im ms bis min Bereich
- Knoten sendet Daten an Coordinator
- **Direct Data Exchange**

Coordinator (CORD) zu End Device (ED)

- Daten an End Device senden
 - End Device ist am schlafen (sleep)
- Coordinator
 - Speichert die Daten
- End Device
 - Wacht periodisch auf
 - Fragt, ob Daten verfügbar sind
 - Wechselt die Polling Rate (Daten Frequenz)



19.1. ZigBee

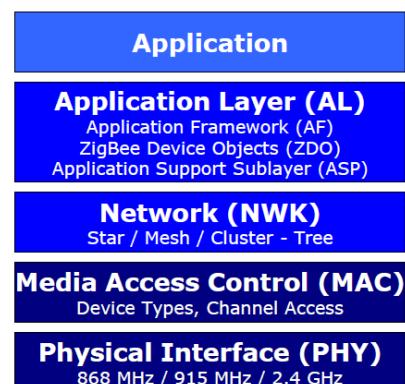
Wieso ZigBee?

IEEE 802.15.4 ist schön und gut, jedoch benötigen wir noch folgendes:

- + Alles oberhalb des MAC Layer
- + Routing
- + Zusätzliche Netzwerktopologien: Mesh (Masche, Geflecht), Cluster-Tree („Büschen-Baum“)
- + Gemeinsam genutzte Netzwerk Infrastruktur
- + Interoperabilität

Was ist ZigBee (Alliance)?

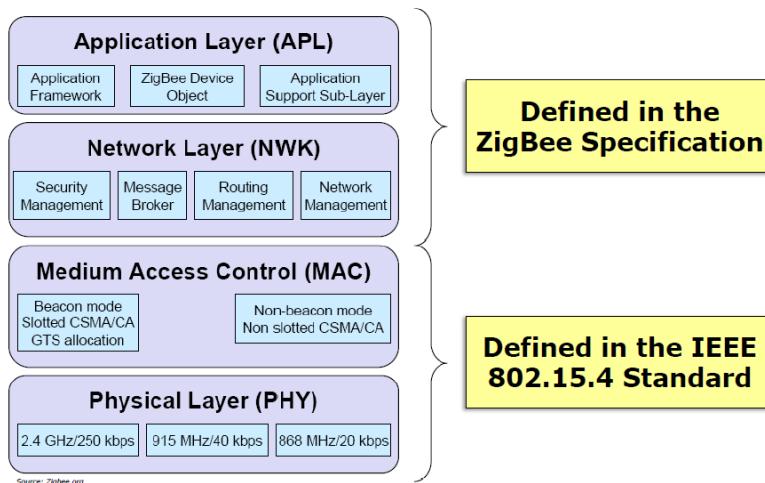
- + (Ist NICHT Open Source. Industrie Konsortium aus End Usern, OEM's, Chip/Software Entwickler)
- + Für Ultra Low Power Wireless Personal Area Network (WPAN)
- + Was macht es anders?
 - Für unterschiedliche Applikationen und Märkte
 - Durchdachtes Netzwerk Management
 - Standardisierung
 - Interoperabilität
 - Low Cost, Low Power
 - Fokus auf niedrige Datenraten
 - Fokus auf Verbindungen mit niedrigem Duty Cycle
- + Architektur?
 - Basiert auf IEEE 802.15.4
 - Beinhaltet AL, NWK, MAC und PHY Layer



Wofür ist ZigBee geeignet?

- + Low Energy Netzwerke (beacon, ...)
- + Monitoring und Systemkontrolle mit niedrigen Datenraten
- + Sporadische Daten, geringer Datenumfang, kleine Pakete
- + Abdeckung von grösseren Gebieten (Mesh Netzwerke)
- Abdeckung von grösseren Gebieten ohne Router
- Grosse Daen Pakete / Datenumfang
- Mobile Applikationen mit häufigen verbinden/trennen – zuordnen/trennen
- Daten Streaming (low quality Audio möglich)

Protokoll Stack: Architektur

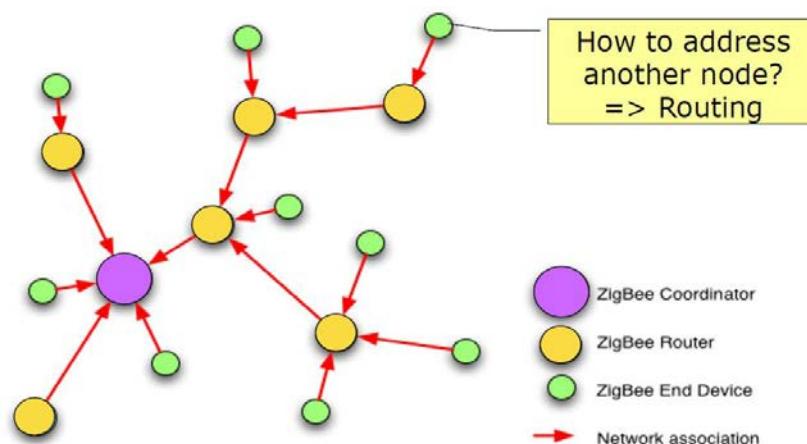


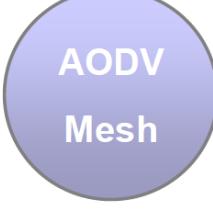
Zwischen den einzelnen Layern sind verschiedene SAP's (Service Access Point) vorhanden.

ZigBee Device Types

- + ZigBee Coordinator (ZC) ≈ PAN Coordinator (FFD), arbeitet optional als Router
 - o Nur ein einziger pro ZigBee Netzwerk
 - o Baut das Netzwerk auf und startet es
- + ZigBee Router (ZR) ≈ Coordinator (FFD)
- + Optionale Netzwerk-Komponente, ordnet sich mit ZC oder existierendem ZR zu
- + ZigBee End Device (ZED) ≈ End Device (RFD)
- + Optionale Netzwerk-Komponente
- + Erlaubt kein Zuordnungen und Routing

Netzwerk Struktur



Routing Methoden		
 Broadcast	 AODV Mesh	 Tree
<p>Node sends packet to all recipients in range. If node is not the final recipient, then it will forward the packet.</p>	<p>Ad Hoc on-Demand Distance Vector Routing): Node sends ROUTE REQUEST (RREQ) to neighbor. The recipient sends a ROUTE REPLY (RREP) in case it is the final destination of the packet, or in case he knows the destination. Otherwise he sends a RREQ.</p>	<p>Node sends packet to parent node. Parent node delivers packet to its own parent node or to his own child node (in case the final destination is a child node).</p>
Profiles		
<ul style="list-style-type: none">• IPM (Industrial Plant Monitoring) = Prozessüberwachung industrieller Systeme<ul style="list-style-type: none">◦ Druck, Temperatur, IR, ...• HA (Home Automation) = Haus/Gebäude oder Büroräume<ul style="list-style-type: none">◦ Licht, Schalter, HVAC (Klima etc.), Wasserpumpen, Storen, Einbrecher Alarmsysteme. ...		

20. Low Power

Leistung (Power) = U*I

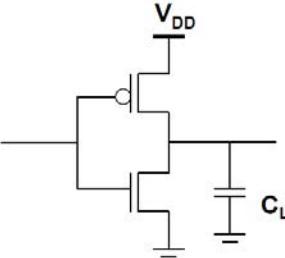
Energie = Integral der Leistung über die Zeit

Zeitfaktor:

- Schneller erledigen: weniger Energie wird benötigt
- Aber es benötigt mehr Energie um etwas schneller zu erledigen

Leistung eines Transistors

$$P_{\text{total}} = \alpha C_L V_{DD}^2 f + t_{sc} V_{DD} I_{\text{peak}} + V_{DD} I_{\text{leak}}$$



P	Power
α	Activity factor
C _L	Transistor capacity
V _{DD}	Supply voltage
f	Clock frequency
t _{sc}	Shortcut time factor
I _{peak}	Shortcut current
I _{leak}	Leakage current

Wie kann die benötigte Leistung gesenkt werden?

External Power

- Low Power Peripherie verwenden
 - z.B. 3.3V anstatt 5V
 - Geräte/Bausteine mit internen Low Power Optimierungen/Shutdown
 - High-End Power Supplies (weniger Verluste in der Speisung)
 - Geräte, welche keine Kühlung benötigen
- Displays
 - Low Power LCD, LED, etc.
 - Displaygrösse verringern
 - Aktivität verringern
 - Helligkeit und Farbe an Umgebungslicht anpassen
 - Zero-Power (b-stabile) Displays
- Schutz vor Umgebungswärme, da der Leakage Current bei Transistoren bei steigender Temperatur auch steigt

Das Ganze ist natürlich auch eine Preisfrage, da qualitativ höherwertige Bausteine meistens auch teurer sind.

Hardware Pins

- Debug Interface
 - Deaktivieren falls nicht benutzt
- Unbenutzte Pins
 - Als Output: Low
 - Als Input: Pull-Ups (internal)
 - Internal Unwired Pins (SiP)

INTRO

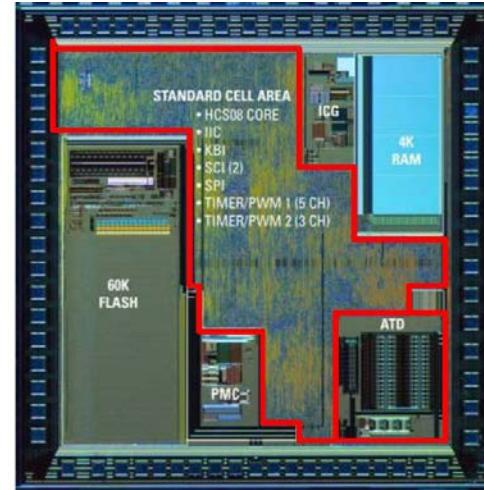
Thomas Rohrer

Page 54 of 69

Einflussfaktoren auf den Stromverbrauch bei einem Mikrocontroller

Die Größe: Memory

- RAM (& Cache)
 - Grosse Fläche
 - Benötigt viel Leistung
- FLASH/EEPROM
 - Geringere Fläche
 - Benötigt weniger Leistung
 - Spezielle Programmierspannung?
- So viel wie benötigt
- ABER: Möglicherweise wird mehr benötigt als man denkt:
 - Stack!
 - Neue Funktionen
 - Produkt Lebenszyklus
 - Entwicklungsunterstützung (trace)

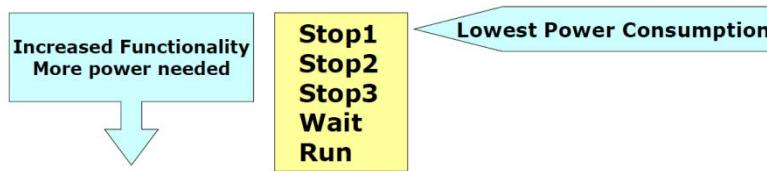


Die Größe: Core & Peripherie

- CPU Core
 - Register
 - Befehlssätze (Instruction set)
 - Pipelines
- Peripherie
 - Timer, PWM, ...
 - I2C, USB, SPI, SCI, ...
 - So wenig wie möglich
- ATD (Analog-to-Digital Konverter)
 - Relativ gross
 - Auf das Minimum reduzieren

Low Power Modes

- Um so weniger Funktionen benötigt werden, desto tiefer kann der Low Power Mode sein
- Wait Mode
 - CPU gestoppt (Clock gating)
 - Bus Clock bleibt aktiv aber wird gesenkt (ca. auf 1MHz)
 - Interrupt: Wait mode wird beendet
 - Vorteile
 - Einfach zu integrieren
 - I_{dd} Reduktion im Vergleich zum Run-Mode
 - Keine Stop Erholungszeit (Interrupts bleiben aktiv)
 - Noise Reduktion für A/D-Wandlung
 - Nachteile
 - Spannungswandler bleiben aktiv
 - Benötigt mehr Energie als in den Stop Modes



Mode	PDC	PPDC	CPU, Digital Peripherals, Flash	RAM	ICG	ATD	Regulator	I/O Pins	RTI
Stop1	1	0	Off	Off	Off	Disabled ¹	Off	Reset	Off
Stop2	1	1	Off	Standby	Off	Disabled	Standby	States held	Optionally on
Stop3	0	Don't care	Standby	Standby	Off ²	Disabled	Standby	States held	Optionally on

¹ Either ATD stop mode or power-down mode depending on the state of ATDPU.

² Crystal oscillator can be configured to run in stop3. Please see the ICG registers.

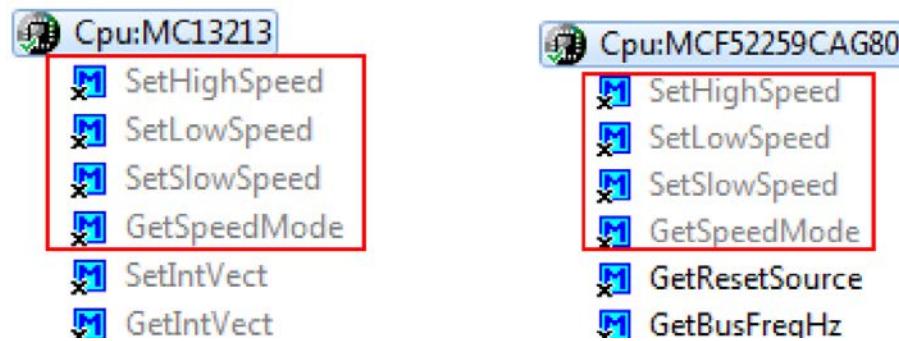
FreeRTOS and Low Power

Die `vApplicationIdleHook()` Funktion kann die CPU in Low Power Modes setzen.

```
70#define configUSE_PREEMPTION           1
71#define configUSE_IDLE_HOOK            1
72#define configUSE_TICK_HOOK           1
73#define configCPU_CLOCK_HZ          CPU_BT
```

Processor Expert

Bei den CPU-Komponenten sind Funktionen verfügbar, um die Low Power Modes zu setzen oder rückzusetzen.



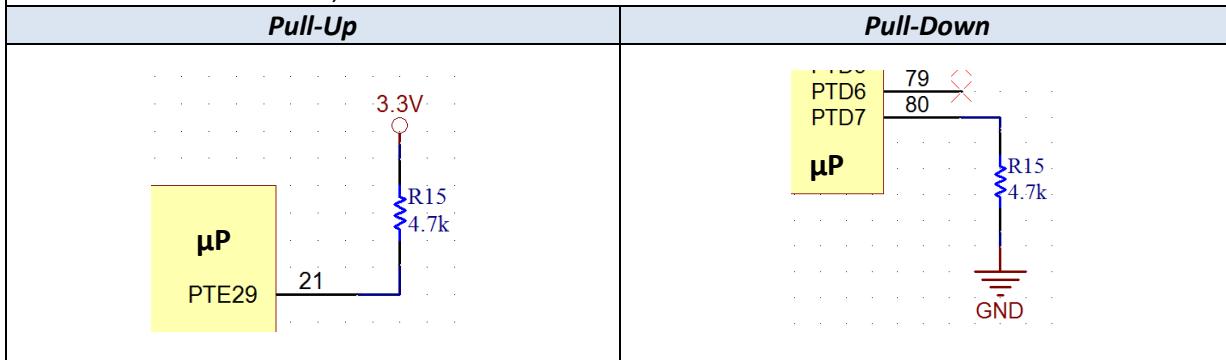
Eventuell müssen auch Timer- und Schnittstellen-Frequenzen angepasst werden (z.B. UART/SCI auf 4800 baud o.ä.).

21. Hardware stuff

21.1. Pull-Up und Pull-Down Widerstände

Wieso werden Pull-Up und Pull-Down Widerstände eingesetzt?

- Mikroprozessoreingänge auf ein definiertes Signal ziehen (vor allem wichtig bei Einschalten/Reset des Mikrocontrollers)
- Werden zum Teil zur Konfiguration von Devices verwendet (Logisch High / Low)
- Open-Drain/Open-Kollektor-Eingänge benötigen meistens einen Pull-Up Widerstand, um ihre Funktion erfüllen zu können
- Open-Source/Open-Emitter-Eingänge (selten vorhanden) benötigen meistens einen Pull-Down Widerstand, um ihre Funktion erfüllen zu können



LED und Vorwiderstand

Bei LED's lieber die Kathode an den Mikrocontroller anschliessen und die Anode mit einem Pull-Up an der Speisung (Bild 1).

Grund: Mikrocontroller erträgt Sink Current meistens besser als Source Current (ich glaube ca. $\frac{1}{2}$ Silizium Die Area für selben Strom).

Bild 2 = Lösung mit praktisch gar keiner Strombelastung für μP

Bild 1

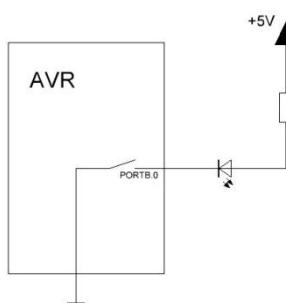
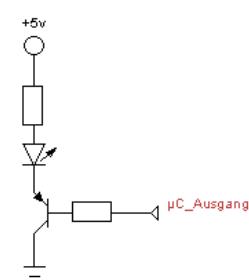
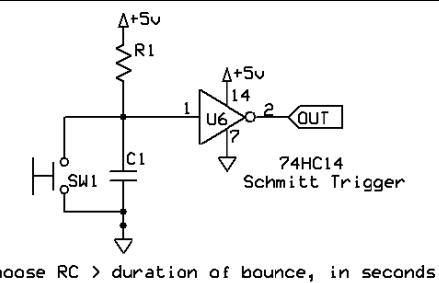


Bild 2



21.2. Entprellen

Entprellen



Wieso?

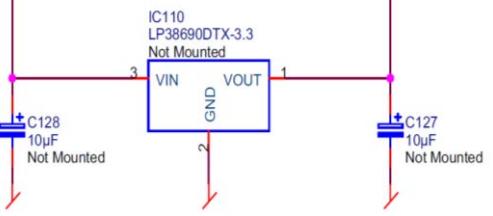
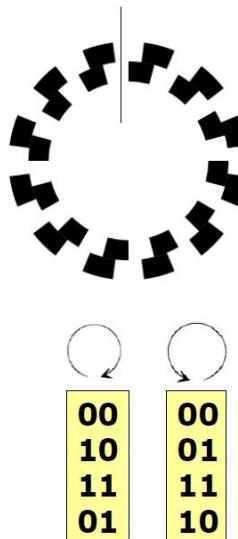
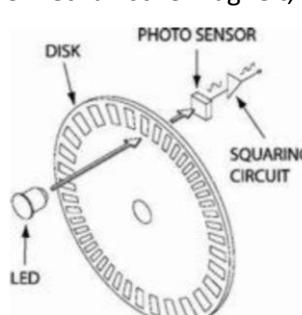
Jeder Schalter prellt!

Funktionsweise

Der Widerstand dient als Pull-Up Widerstand. Durch den Kondensator werden allfällige Peaks gedämpft. R und C so wählen, dass die Zeitkonstante $R \cdot C$ grösser als die Prelldauer ist (sinnvoller Wert meistens ca. 20-40ms).

21.3. Verschiedene Devices / Bausteine

RS-232 Treiber / Level-Shifter	Quarze und Oszillatoren
<p>Funktionsweise Ein RS-232 Level Shifter ändert wie der Name schon sagt die Signalpegel. Zwischen µP und Level Shifter beträgt die Spannung TTL Niveau (3.3V/5V o.ä.) und nach dem Level Shifter normalerweise ±9...15V. Dadurch erhält man eine bessere Störfestigkeit (SNR wird grösser).</p>	<p>Funktionsweise Ein Oszillator oder Quarz erzeugt einen Clock. Von diesem externen Referenzclock ausgehend können nun die µC-Clocks (CPU, Bus etc.) generiert werden. Es kann bei vielen µP's auch ein interner Oszillator verwendet werden, jedoch ist seine Frequenz meistens ungenauer als die eines externen Oszillators.</p>
<p>H-Brücke</p>	<p>Eine H-Brücke kann zur Steuerung eines Motors verwendet werden. Dabei kann der Motor in beide Drehrichtungen betrieben werden (blau = z.B. vorwärts / grün = rückwärts)</p> <p><u>Idee:</u></p> <ul style="list-style-type: none"> • 4 Schalter (Transistoren) • Jeder individuell ansteuerbar <p><u>Anforderungen:</u></p> <ul style="list-style-type: none"> • Exaktes Timing • Schalter müssen synchron schalten <p><u>Motortreiber (Signale):</u></p> <ul style="list-style-type: none"> • Drehrichtung • PWM (für Geschwindigkeit) • Andere (Nothalt, etc.)
<p>A/D Wandler (ADC)</p> <p>Ein A/D-Wandler oder ADC wandelt einen analogen Wert in einen digitalen Wert um. Folgend ein paar Eigenschaften eines ADC:</p> <ul style="list-style-type: none"> • Resolution in Bit, 12 Bit für µP sehr häufig • Das Wandlungsverfahren in einem µP ist meistens „sukzessive Approximation“ • Full Scale Wert wird von einer Referenzspannung vorgegeben (z.B. 3.3V) • Eine kürzere Wandlungszeit (in Samples/s) bedeutet meist auch eine ungenauere Messung 	<p>3-Axis Accelerometer (Beschleunigungssensor)</p> <p>Funktionsweise Generiert aus der Beschleunigung in allen 3 Dimensionen (X, Y, Z) drei Spannungen. Diese können vom µC mittels A/D-Wandler eingelesen werden.</p>

Spannungsregler	Quadratur Encoder
 <p>Funktionsweise Generiert an seiner höheren Spannung bei „VIN“ eine tiefere Spannung bei „VOUT“ (z.B. 3.3V). Die Beiden Kondensatoren dienen als Filterung/Stützung der Speisung.</p>	<p>Funktionsweise Wandelt einen Winkel in ein digitales Signal um (z.B. bei Motoren). Dabei sind die 2 Signale (innerer und äusserer Ring) im Gray-Code codiert (nur ein einziger Bitwechsel beim nächsten Zustand).</p> <p>Informationen:</p> <ul style="list-style-type: none"> • Geschwindigkeit • Drehrichtung <p>Ausführungen:</p> <ul style="list-style-type: none"> • Mechanisch <ul style="list-style-type: none"> ◦ prellen, schlecht für high speed • Magnetisch <ul style="list-style-type: none"> ◦ Hall Sensoren, für raue Umgebungen • Optisch <ul style="list-style-type: none"> ◦ Keine mechanische Trägheit, High speed  
Bloons Tower Defense Sensor	
 <p>Funktionsweise Reagiert auf alle Arten von Bloons. Höchst empfindlich und <u>überlebenswichtig!!!</u></p>	

22. Compiler and Linker stuff

Seite 455 ff. im Skript

Kapitel = Understanding Your C Compiler: How to Minimize Code Size

The Structure of a Compiler

In general, a program is processed in six main steps in a modern compiler (not all compilers follow this blueprint completely, but as a conceptual guide it is sufficient):

- **Parser:** The conversion from C source code to an intermediate language.
- **High-level optimization:** Optimizations on the intermediate code.
- **Code generation:** Generation of target machine code from the intermediate code.
- **Low-level optimization:** Optimizations on the machine code.
- **Assembly:** Generation of an object file that can be linked from the target machine code.
- **Linking:** Linking of all the code for a program into an executable or downloadable file.

23. Laboratory Short Courses

23.1. Exploring Embedded C

- Find the memory map of your microcontroller and list the address ranges of RAM and ROM
 - Siehe "Memory Map" des Datenblattes
- What data types does your compiler support and how many bits of storage are required for each?
 - char 8 Bit
 - short int 16 Bit
 - long int 32 Bit
 - float 32 Bit
 - double 64 Bit
 - "int" ist generell abhängig vom Zielsystem (vgl. 8-Bit oder 32-Bit CPU)

Variables

- What does "scope" of a variable mean?
 - Bereich, wo die Variable gültig ist, d.h. wo man sie „sieht“
- A variable is defined inside a function and the compiler does not allocate a specific memory location in RAM for its storage. a) Is this an automatic or static variable? b) Where is it stored? c) What is its scope? d) Is data stored in this variable during one function call available to the function in the next call?
 - a) automatic
 - b) stack
 - c) nur innerhalb der Funktion
 - d) nein
- A static variable is defined inside a function. a) What is its scope? b) Where is storage space allocated for it? c) Is data stored in this variable during one function call available to the function in the next call?
 - a) nur innerhalb der Funktion
 - b) RAM
 - c) ja
- A variable is defined outside a function. a) Is this a static or automatic variable? b) If another separately compiled function wishes to use this variable, what must be done in this function to do this?
 - a) static
 - b) sie muss als „volatile“ deklariert werden

Minimal Startup

- With minimal startup code for the S08, you should get an error or warnings when you make the project? What does it mean?
 - Warning message: "Initializaton data lost." → Initialisierungswerte wurden überschrieben oder gar nicht erst beschrieben
- What changes must you make to your program to compensate for the problem?

- Da der minimale Startup Code nur den Stackpointer initialisiert, müssen Sie sich um allfällige Initialisierung von globalen Variablen selber kümmern; z.B. in einer Init() Routine.

Plain Char

- For your HCS08 compiler, is 'char' a signed or unsigned type?
 - CodeWarrior 10.3: signed

Using Individual Bits in a Memory Location

- Does your compiler produce bit addressing instructions such as BSET and BCLR?
 - Yes ???

CodeWarrior I/O Port and Bit Addressing

- What does the following line of code accomplish?

```
extern volatile PTFDSTR PTFD @0x00000040;
```

 - Port D @ Adresse 0x00000040 wird als „flüchtig“ deklariert
- Explain how the structure PTFDSTR works to allow you to access PTFD as a byte or as individual bits in the port.
 - Annahme: Es wird immer das ganze Byte beschrieben. Mit Masken könne jedoch auch nur einzelne Bits beschrieben werden.

Using Interrupts

- What makes an interrupt handler or service routine different than an "ordinary" function?
 - Eine ISR wird nur auf externen Abruf ausgeführt. D.h. sie wird nicht vom Programmcode selber direkt aufgerufen, sondern von einem externen Event wie z.B. einer steigenden Flanke am Pin XY.

Connecting a C Program to an Assembly Language Program

- Does your compiler allow you to insert assembly language instructions "in-line" with your C code? If so, how do you do this?
 - Ja. Mit dem Keyword "asm {xy_code};" kann Code eingefügt werden.
- How does your C program transfer arguments to and from the assembly language program?
 - Es wird so oft es geht über die Register der CPU gearbeitet und möglichst wenig über den Stack

23.2. Serial I/O Interfaces – RS-232-C

Explore 2

- What part would you chose to include in an embedded system design to convert from the CMOS logic levels of the SCI to RS-232-C and provides at least two input and two output signals?
 - MAX3232CSE+ (3...5.5V, 2 Treiber)

Stimulate 2

- Write a programming algorithm (a design) for a DCE device that allows it to transmit data only when another device (DTE or DCE) is ready for it.
 - Set CTS
 - Read RTS
 - If RTS ok send, else repeat loop
- Write a programming algorithm (a design) for a DTE device that allows it to transmit data only when another device (DTE or DCE) is ready for it.
 - Set RTS
 - Read CTS
 - If CTS ok send, else repeat loop

23.3. Interrupts using C

- Can you give three examples of I/O devices for which interrupts could be used to synchronize the microcontroller with the I/O?
 - Push Button; Handshaking bei Kommunikationsschnittstellen (z.B. SPI); externe Events wie z.B. fallende Flanken oder Brown Out
- Assume you are designing microcontroller systems to be used in an automotive application. Give three examples of important events that would lend themselves to being implemented in a microcontroller using an interrupt system.
 - Temperatur überschritten; Drehzahlmessung eines Motors (Fächerscheibe / Lochscheibe); Prozesskontrolle wie z.B. Schritt 3 von 5 erreicht

Explore 1

- Where in your documentation do you find the vector locations (or vector addresses) for interrupts?
 - Im Datenblatt unter MCU Resets, Interrupts etc. (Kapitel 12.5)
- How does your microcontroller allow for asynchronous events to occur and be recognized?
 - Mittels Prioritäten kann die Reihenfolge der ISR's eingestellt werden. Tritt ein Interrupt auf, wird das Interrupt Flag gesetzt.
- How does your microcontroller branch to the correct interrupt service routine?
 - Beim Interrupt wird die Adresse der ISR (Vektor-Adresse) in den Program Counter (PC) geladen und somit als nächstes diese Adresse angesprungen.
- How does your microcontroller return to the interrupted program at the point it was interrupted?
 - Vor dem Sprung in die ISR wird die nächste Adresse im Program Counter (PC) auf den Stack geladen und am Schluss der ISR wieder in den Program Counter zurückgeladen.
- How does your microcontroller allow the programmer to globally enable and disable all interrupts?
 - In einem Register wird das allgemeine Interrupt Enable Flag gesetzt.
- How does your microcontroller allow the programmer to enable and disable selected interrupts?
 - Mit der "Interrupt Mask" können einzelne Interrupts enabled oder disabled werden (in einem Register werden Bits gesetzt)
- How does your microcontroller disable further interrupts so the first can be serviced without being interrupted?
 - Mittels Prioritäten oder "Enter Critical"
- How does your microcontroller deal with multiple, simultaneous interrupts?
 - Führt die Interrupts je nach Priorität aus.
- What can cause a pending interrupt?
 - Wenn schon ein interrupt ausgeführt wird, wenn der IRQ kommt
- How do you reset a flag that caused an interrupt?
 - Am besten am Anfang der ISR unter Berücksichtigung der Prozessoreigenschaften.
- What happens if you do not reset the flag in the interrupt service routine?
 - Der Interrupt wird immer wieder in einer Endlosschlaufe ausgeführt.

Stimulate 2

- There is always a delay between the interrupt request and when the CPU starts to execute the interrupt service routine. This is called the interrupt latency. Give three components that cause interrupt latency.
 - Interrupt asynchron zu clock; CPU führt aktuellen Befehl zu Ende aus; evtl. Register sichern; evtl. ist schon ein anderer Interrupt mit selber oder höherer Priorität am ausführen
- Give an advantage and a disadvantage of automatically pushing registers onto the stack?
 - Pro: Definierter Zustand nach Zurückspringen aus der ISR; muss nicht selbergemacht werden
 - Contra: Nicht unbedingt bei jedem Interrupt nötig (benötigt Zeit und Speicher)

Explore 4

- How do you signify to the compiler that a function should be treated as an interrupt handler or service routine?
 - **ISR(XY_Interrupt) {Put ISR Code here}**

23.4. Analog Input Sampling

Stimulate 1

- For an A/D with 8-bits and a 5 volt maximum (full scale) input, what is the smallest change in the input signal that can be detected?
 - 19.6mV → $5V * [1/(2^n - 1)]$
- For an A/D with 10-bits and a 5 volt maximum (full scale) input, what is the smallest change in the input signal that can be detected?
 - 4.88mV

Stimulate 2

- What elements affect the conversion time for various A/D converters?
 - Resolution, Wandlungsverfahren, System clock frequency
- Your microcontroller's A/D may have its own clock derived from the system clock. How is the A/D clock frequency determined?
 - System clock frequency und über ein Register (z.B. Clock divider)

Stimulate 3

- What does bandwidth limited mean?
 - Frequenzbereich ist begrenzt (keine unendlichen Frequenzen!)
- What kind of electronic filter must you use to limit the bandwidth of a signal?
 - Tiefpass
- You wish to digitize a signal whose maximum frequency is 1 kHz with your A/D.
 - What is the minimum sample rate (in samples/second) that can be used?
 - 125 (Sukzessive Approximation) → 1kHz / n
 - What is the maximum conversion time?
 - 8ms → 1s / (samples/second)

Stimulate 4

- $\text{Dynamic Range} = \frac{V_{MAX}}{V_{NOISE}}$
- What is the dynamic range of a signal whose maximum is 26 volts and whose noise is 26 mV?
 - 1000
- Dynamic range is often expressed in decibels. What is the dynamic range of the signal above in decibels?
 - 60dB → $10^{\left(\frac{xx\ dB}{20}\right)} = 1000$

Stimulate 5

- Consider a sinusoidal signal $v(t)$ where V_{MAX} is 5 volts and f_{MAX} is 1 kHz. The aperture time of the A/D is 1μs. What is the worst-case change in voltage, as a percent of full scale that can occur? → $\text{Analog Input (Sinus)} = v(t) = V_{MAX} * \sin(2\pi * f_{MAX} * t)$
 - $\Delta v(t) = v(1\mu s) - v(0) = 0.55mV$
 - $\Delta v(t) \text{ in \%} = \frac{0.55mV}{5V} * 100 = \underline{\underline{0.011\%}}$

Explore 3

- What influences the aperture time in your microcontroller?
 - Wie schnell der „Sample and Hold“-Block ist; Wandlungszeit

Auswahlkriterien für A/D-Wandler

1. $2^n \leq \frac{V_{MAX}}{V_{NOISE}}$
2. $2^n \geq \frac{V_{MAX}}{V_{Smallest_Signal}}$
3. $Conversion\ Time < \frac{1}{2*f_{MAX}}$
4. $Sampling\ Frequency > 2 * f_{MAX}$
5. $Aperture\ Time\ t_{AP} \leq \frac{1}{2\pi*f_{MAX}*2^n}$

Stimulate 6

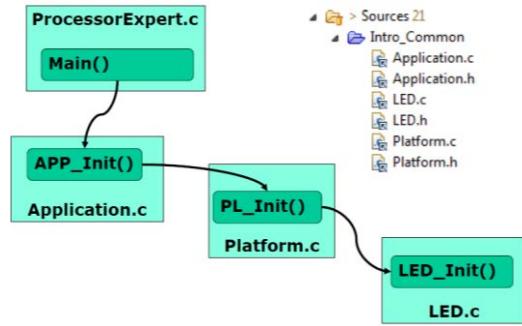
- A temperature transducer produces an analog voltage from 0 to 5 volts over a range of temperatures from 0 to 100°C. When an oscilloscope is used to look at the output of the transducer, we see a noise level of about 1 mV peak-to-peak. How many bits are needed in the A/D so that the electronic noise is less than the quantization level?
 - 1 bis 11 Bits (Formel 1)
- The same temperature transducer is being used to provide a temperature display of 0 to 100°C with a resolution of 1°C. How many bits are needed in the A/D for this case?
 - 7 Bit ($2^7 = 128 > 100$)
- You wish to digitize a 100 kHz signal.
 - What is the minimum sample frequency?
 - $200\text{kHz} = 2*f_{Signal}$
 - What is the maximum conversion time?
 - $5\mu\text{s} = 1/f_{Sample_Minimum}$
- You look up the specifications for an A/D and find its conversion time is 16μs. What is the maximum frequency that can be converted?
 - $31.25\text{kHz} \rightarrow 16\mu\text{s} = \frac{1}{2*f_{MAX}} \rightarrow f_{MAX} = \frac{1}{2*16\mu\text{s}} = 31.25\text{kHz}$

Stimulate 9

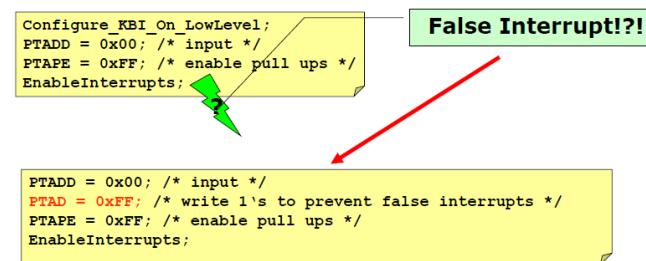
- Propose a circuit using two diodes that will limit the voltage at the input to the A/D converter to at most one diode drop above the maximum input and one diode drop below the minimum.
 - Clamping Dioden (1 Diode mit Kathode gegen Speisung; 1 Diode mit Anode gegen GND)

24. Sonstiges

Application Initialization Flow



Keys



24.1. Stichwörter Example Exam A

Folgende Themen kommen im Teil A der Example INTRO MEP vor:

- Realtime
- Interrupts / Timing / ISR
- Datentypen / Grösse
- Doxygen
- Eclipse / Compiler / Embedded C
- Makros
- Reentrant
- Synchronisation
- Critical Sections
- Code Verständnis
- State Machine / Mealy
- Hardware (Pullup / Pulldown; IC's)

24.2. Stichwörter Example Exam B

Folgende Themen kommen im Teil B der Example INTRO MEP vor:

- Realtime and its requirements
- Processor Expert
- VCS (Version Control System) à la SVN
- Makros
- Synchronisation
- Doxygen

- Shell
- Polling and Interrupts
- FreeRTOS (Open Source)
 - Tick Timer
 - Pre-emptive und cooperative multitasking (RTOS)
 - Semaphores
 - Priority Ceiling
 -
 - Task Timing
 - Task Schedule
 - Startup
- stdin, stdout, stderr
- Hardware
 - Internal Pull-Up
 - RS-232 Level Shifter
 - H-Bridge Ansteuerung
 - Oscillator / Crystal
 - Digital Encoder
 - A/D-Converter
 - Accelerometer
 - Debounce
 - Quadratur Encoder
- ANSI C
 - strcmp() vs. strncmp()
 - enum vs. #define)
- Debounce
- ZigBee Network
- Low Power Application
 - HCS08 Properties und Einstellungen
 - LEDs
 - Hardware Development
 - Software Development
- Code Verständnis
- Netzwerk (IEEE802.15.4)
- Gray Code
- SCI (RS-232)
- Queues
- PWM
- Regler
 - Differenzengleichung Code → Reglerart
- Compiler
- C++
 - Overloading
- State Machine
- Communication Interfaces (SPI, I2C, SCI, USB, „RS-232“)
- Trigger
- Events