Berner Fachhochschule
Haute école spécialisée bernoise
Bern University of Applied Sciences

Hardwarenahe Softwareentwicklung

Exceptions und Interrupts

V5.1, ©2023 roger.weber@bfh.ch

Lernziele

Sie sind in der Lage:

- Die Mechanismen beim Auftreten einer Exception zu erklären.
- Hardware-Peripherie, Exception Handler und Interrupt-Service-Routinen korrekt zu implementieren.

Inhaltsverzeichnis

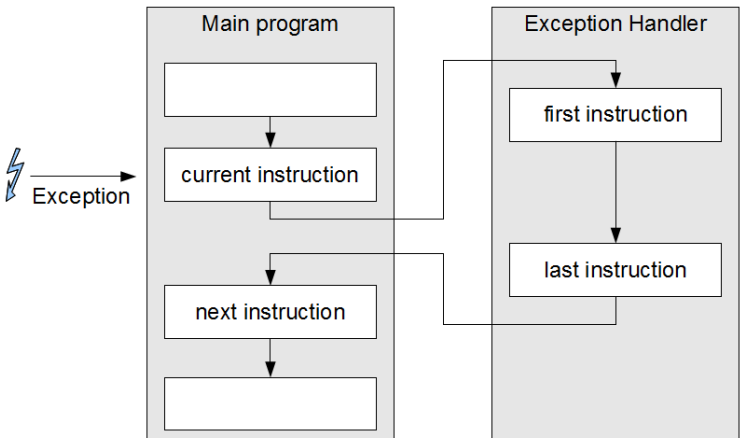
Einführung

Exceptions

- Exceptions sind Ereignisse, welche während der Ausführung eines Programms auftreten und eine Ausnahmebehandlung (Excepiton Handler) erfordern.
- Das Eintreffen einer Exception ist **asynchron** zum Programmablauf (Ausnahme: Software-Interrupt).

Was für Ereignisse können in einem Embedded System einen Interrupt auslösen?

Unterbruch des laufenden Programms durch Exceptions



Quellen von Exceptions [1]

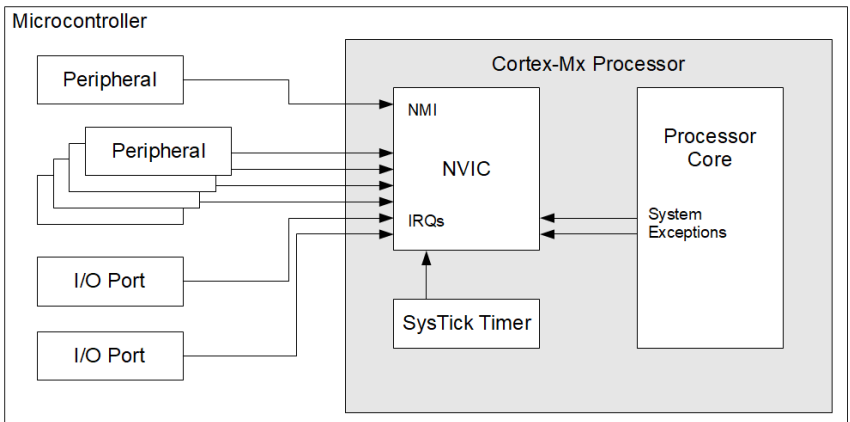
- **System Exceptions**, werden durch die CPU erzeugt.
  Beispiele:
  - NMI (Non Maskable Interrupt)
  - Memory Management Fault
  - HardFault
  - BusFault
- **Hardware-Interrupts**, werden durch die HW-Peripherie erzeugt (**asynchron**).

Merke

→ Interrupts sind ein wichtiges Hilfsmittel, um Echtzeitanforderungen zu erfüllen.

- **Software-Interrupts**, werden durch spezielle Assembler-Instruktionen ausgelöst (Betriebssystem-Aufruf, **synchron**).

Quellen von Exceptions [2]



Maskierung

- Interrupts können gesperrt (maskiert, disabled) und freigegeben werden (enabled).
- Beim Cortex-Mx erfolgt die Maskierung dreistufig:
  1. Global
     - PRIMASK (ausser Reset, NMI, HardFault)
     - BASEPRI (Exceptions unter einer definierten Priorität)
  2. NVIC, jede Vektoradresse
  3. Jede Interrupt-Quelle individuell

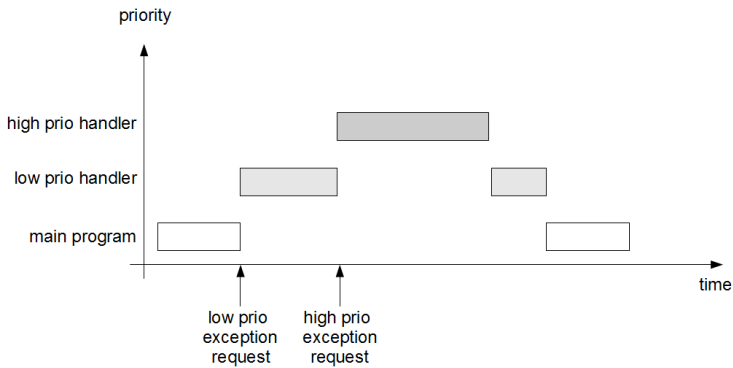
Globale Maskierung der Interrupts

Maskierung der Interrupts mit Hilfe der in CMSIS definierten inline Funktionen __enable_irq() und __disable_irq().

```
/**
\brief   Enable IRQ Interrupts
\details Enables IRQ interrupts by clearing the I-bit in the CPSR.
Can only be executed in Privileged modes.
*/
__STATIC_FORCEINLINE void __enable_irq(void)
{
    __ASM volatile ("cpsie i" : : : "memory");
}


/**
\brief   Disable IRQ Interrupts
\details Disables IRQ interrupts by setting the I-bit in the CPSR.
Can only be executed in Privileged modes.
*/
__STATIC_FORCEINLINE void __disable_irq(void)
{
    __ASM volatile ("cpsid i" : : : "memory");
}
```

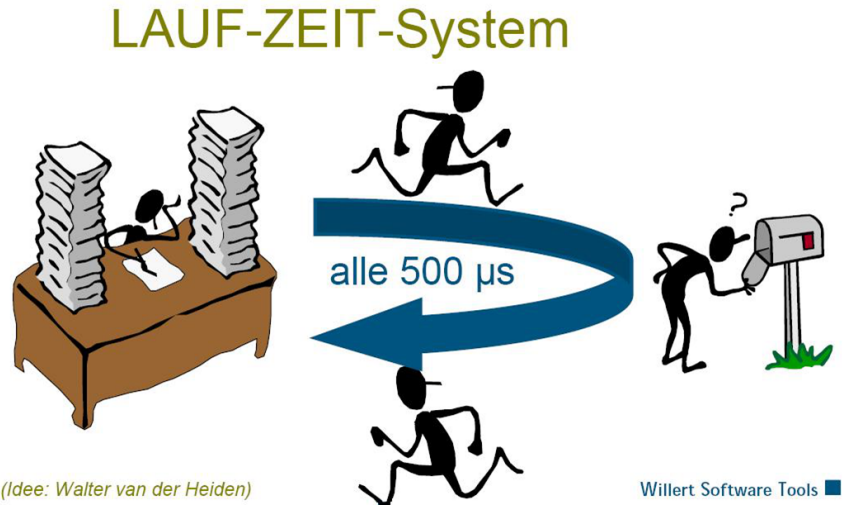
Priorisierung

- Vergabe von Prioritäten
- Eine Exception / ein Interrupt hoher Priorität kann eine Exception / einen Interrupt tiefer Priorität unterbrechen.



Vergleich gepollte / interruptgesteuerte Systeme

Gepollte Systeme [1]



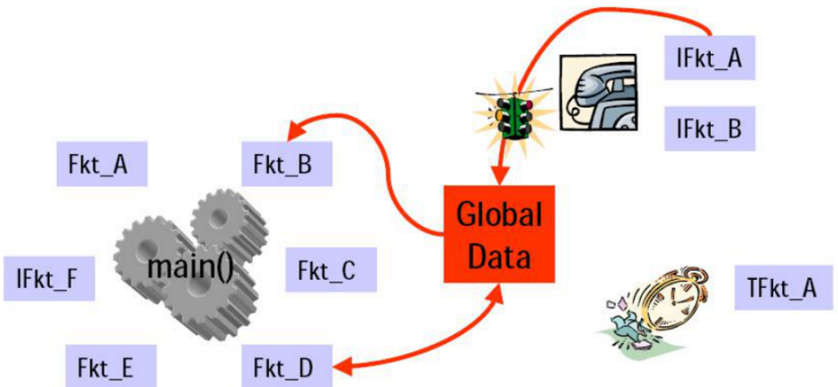
Gepollte Systeme [2]

- Vorteile:
  - Einfache Struktur
- Nachteile:
  - Es wird viel Rechenzeit verschwendet, um die Eingänge dauernd zu pollen.
  - Ist die Zeit zwischen zwei Poll-Zyklen zu lange, kann es sein, dass ein Ereignis verpasst wird.
  - Die Zeit die verstreicht, bis auf das Ereignis reagiert wird, ist im "worst case" gleich der Zeit, die zwischen zwei Poll-Zyklen liegt.

Interrupt-gesteuerte Systeme

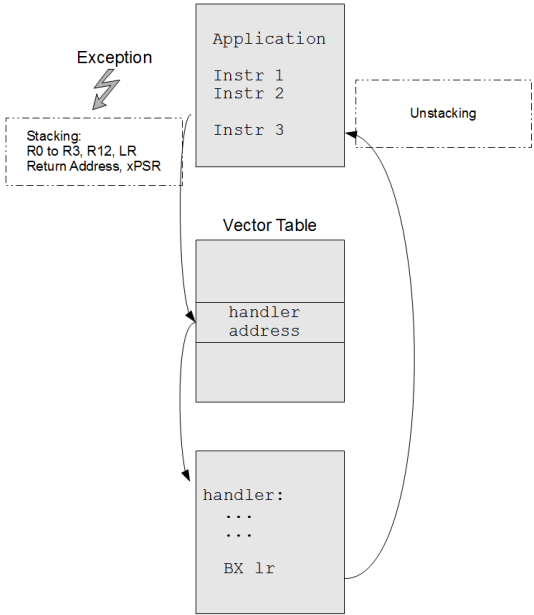
- Vorteile:
  - Sehr schnelle Reaktion auf Ereignisse.
  - Belastung der CPU nur falls wirklich eine Reaktion erforderlich ist.
- Nachteile:
  - Komplexere Design-Struktur.
  - Datenaustausch zwischen Exception Handler und Applikation.

Datenaustausch Exception Handler ↔ Applikation



Ablauf einer Exception-Anforderung

Übersicht

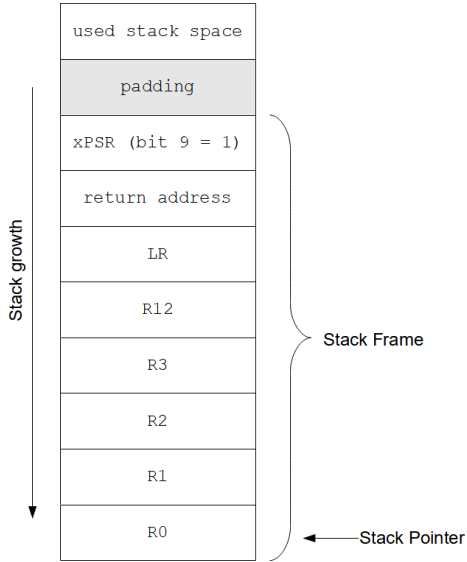

Application
Instr 1
Instr 2
Instr 3
Exception
Stacking:
R0 to R3, R12, LR
Return Address, xPSR
Unstacking
Vector Table
handler
address
handler:
...
...
BX lr

Stacking

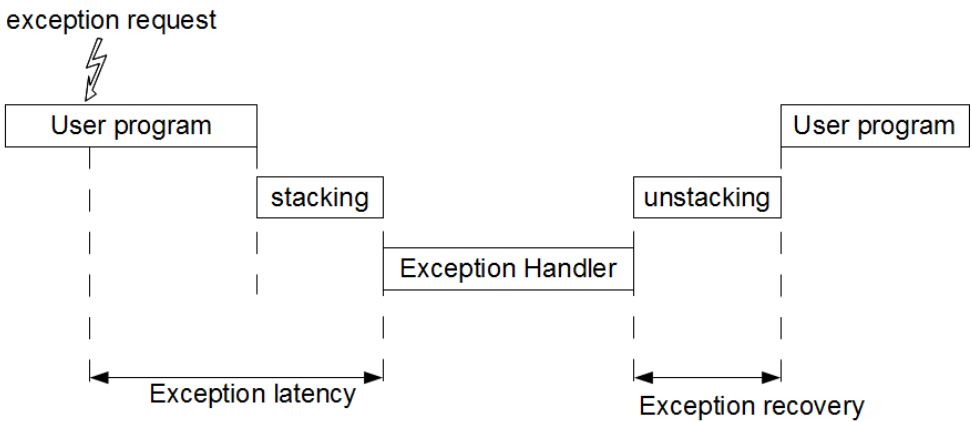
- Folgende Register werden automatisch (durch die CPU-Hardware) auf dem Stack gerettet:
  - r0 bis r3, r12
  - lr und Rücksprungadresse
  - xPSR

Vorteile

→ Einfachere Programmierung der Exception-Handler



Zeitliches Verhalten



Die Vektortabelle

Vektortabelle des Cortex-Mx [1]

- **Vektoradresse**: 1 Exception- oder Interrupt-Quelle, Eintrag der Adresse des Exception-Handlers, welcher die Exception bearbeitet.
- **Vektortabelle**: Tabelle von Vektoradressen, 256 Einträge.
- Für Cortex-Mx:
  - Position 0: Initialisierungswert Main Stack-Pointer
  - Position 1 – 15: System Exceptions
  - Ab Position 16: Interrupts

Vektortabelle des Cortex-Mx [2]

Exception Number	CMSIS Interrupt Number	Table Address Offset	Exception Type	Priority	Description
-	-	0x00	-	-	Initial value of Main Stack-Pointer
1	-	0x04	Reset	-3(höchste)	Reset
2	-14	0x08	NMI	-2	Non-maskable Interrupt
3	-13	0x0C	HardFault	-1	Classes of fault when handler cannot be activated
4	-12	0x10	MemManage	programmable	Memory Management fault
5	-11	0x14	BusFault	programmable	Bus-Sytems fault
6	-10	0x18	Usage Fault	programmable	Invalid instruction or state transition
7 - 10	-	0x1C - 0x28	-	-	reserved
11	-5	0x2C	SVC	programmable	supervisor call
12	-4	0c30	Debug Monitor	programmable	software based debug
13	-	0x34	-	-	reserved
14	-2	0x38	PendSV	programmable	pendable request for system service
15	-1	0x3C	SysTick	programmable	System Tick Timer
16	0	0x40	IRQ	programmable	IRQ Input # 0
...	...	...	...	...	IRQ Input # n
255	239	0x3FF	IRQ	...	IRQ Input # 239

Implementation der Vektortabelle

Auszug Code der Vektortabelle in der Datei startup.s

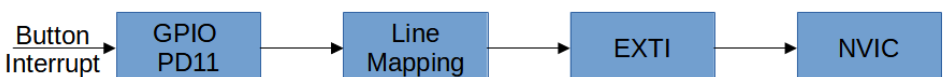
```
/* Definition Vector table */
g_pfnVectors:
    .word   _estack
    .word   Reset_Handler
    .word   NMI_Handler
    .word   HardFault_Handler
    .word   MemManage_Handler
    .word   BusFault_Handler
    .word   UsageFault_Handler
    ...
    .word   EXTI15_10_IRQHandler        /* External Line[15:10]s    */

/* Definition of default handler, example EXTI15_10 */
    .weak       EXTI15_10_IRQHandler
    .thumb_set  EXTI15_10_IRQHandler, Default_Handler
```

- **thumb_set**: This performs the equivalent of a .set directive in that it creates a symbol which is an alias for another symbol. This directive also has the added property in that it marks the aliased symbol as being a thumb function entry point.
- **weak**: The weak attribute causes the declaration to be emitted as a weak symbol rather than a global. This is primarily useful in defining library functions which can be overwritten in user code.

Interrupts auf dem Cortex-Mx

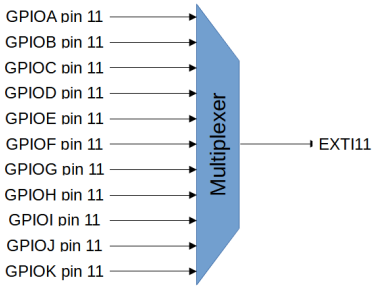
Übersicht

- Am Beispiel des Tasters ganz rechts auf dem Leguan-Board:
  - Der Taster ist mit GPIO Port D Pin 11 (PD11) verbunden.
  - Line Mapping auf EXTI line 11
  - Im NVIC auf EXTI15_10_IRQHandler



Interrupt Line Mapping

- Multiplexer für verschiedene Quellen
- Programmiert in Register SYSCFG_EXTICRx
- Pin 11: EXTI11[3..0] in SYSCFG_EXTICR3, pin 12 bis 15



12.3.4 SYSCFG external interrupt configuration register 3 (SYSCFG_EXTICR3)

Address offset: 0x10

Reset value: 0x0000 0000

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
EXTI11[3:0]				EXTI10[3:0]				EXTI9[3:0]				EXTI8[3:0]			
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

[Quelle: RM0433 Reference Manual STM32H7xx von ST Microelectronics]

Programmierung Line Mapping in C

Auf einem STM32 können die EXTICR-Register für das Line Mapping auf folgende Arten initialisiert werden:

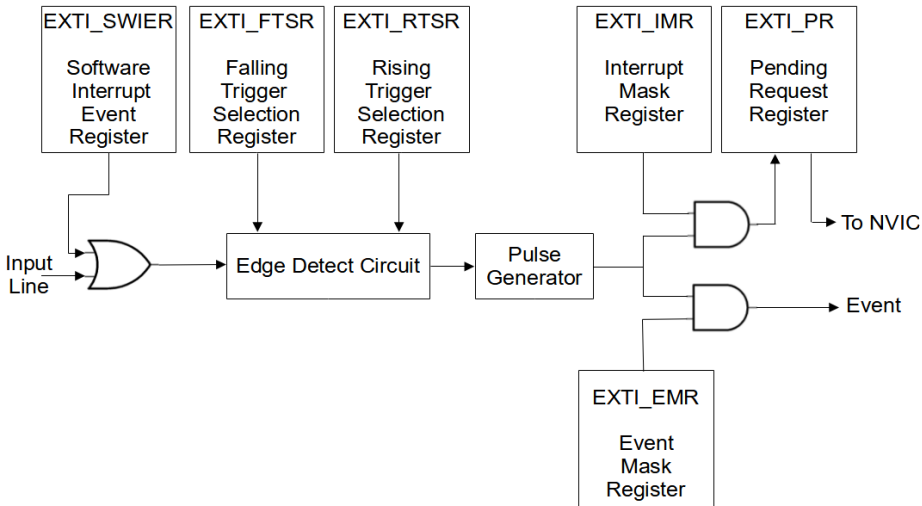
- Durch Angabe der einzelnen Bits

```
/* Multiplexer for EXTI line 11 is set to GPIOD */
/* EXTICR3 is EXTICR[2] */
SYSCFG->EXTICR[2] &= ~(0xF000);    // clear all pins (12 to 15) for EXTI11
SYSCFG->EXTICR[2] |= 0x3000;     // set pins to 0011 (bit 12 and 13) for
    line mapping to port D
```

- Durch Verwendung von #defines aus stm32h743xx.h von STMicroelectronics

```
SYSCFG->EXTICR[2] &= ~(SYSCFG_EXTICR3_EXTI11_Msk);
SYSCFG->EXTICR[2] |= SYSCFG_EXTICR3_EXTI11_PD;
```

EXTI (External Interrupt/Event Controller)



EXTI Register

Register	CMSIS	Function
Interrupt Mask Register	EXTI→IMR	Write 1 to activate
Event Mask Register	EXTI→EMR	Write 1 to activate
Rising Trigger Selection Register	EXTI→RTSR	Write 1 to enable rising edge
Falling Trigger Selection Register	EXTI→FTSR	Write 1 to enable falling edge
Software Interrupt Event Register	EXTI→SWIER	Write 1 to generate interrupt request
Pending Register	EXTI→PR	1 if trigger request occured

Papierübung EXTI Register

Für den Taster des Leguan-Boards (PD11) sollen die EXTI-Register so programmiert werden, dass bei steigender Flanke ein Interrupt generiert wird. Infos zu den EXTI-Registern finden Sie im Dokument “RM0433 Reference Manual STM32H7xx“ von ST Microelectroinc.

- Welche Register müssen Sie initialisieren?
- Welche Bits in den Registern müssen Sie setzen/löschen?

NVIC (Nested Vectored Interrupt Controller)

- Aufgabe: Verarbeitung der Interrupts
- Bis zu 240 Interrupt-Inputs, NMI, 15 System Exceptions
- Priorisierung der Quellen
- Maskierung der Quellen
- Nested: Verarbeitung von verschachtelten Interrupts
- Vectored: Eintrag für jeden Handler (Vektoradresse)

NVIC Register

Register	CMSIS-Core	Function
Interrupt Set Enable	NVIC→ISER[0] to NVIC→ISER[7]	Write 1 to set enable
Interrupt Clear Enable	NVIC→ICER[0] to NVIC→ICER[7]	Write 1 to clear enable
Interrupt Set Pending	NVIC→ISPR[0] to NVIC→ISPR[7]	Write 1 to set pending status
Interrupt Clear Pending	NVIC→ICPR[0] to NVIC→ICPR[7]	Write 1 to clear pending status
Interrupt Active Bit	NVIC→IABR[0] to NVIC→IABR[7]	Active status bit, read only.
Interrupt Priority	NVIC→IP[0] to NVIC→IP[239]	Level (8 bit) for each interrupt

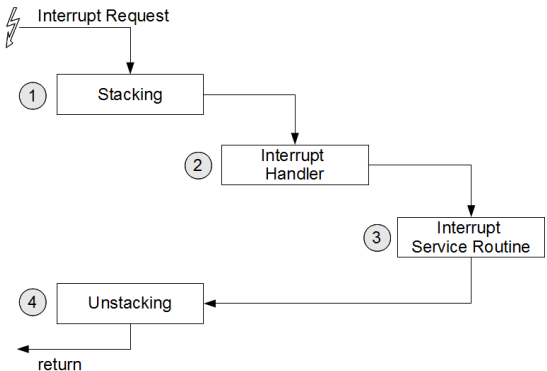
Interrupt Handler

Varianten

- Es gibt verschiedene Varianten, einen Interrupt-Handler zu implementieren:
  - “Nonnested Interrupt Handler“
  - “Nested Interrupt Handler“
  - “Reentrant Interrupt Handler“
  - “Prioritized Simple Interrupt Handler“

Nonnested Interrupt Handler

- Eigenschaften
  - Interrupts werden sequenziell verarbeitet. Während der Bearbeitung eines Interrupts kann kein weiterer Interrupt gleicher Priorität bearbeitet werden.
  - Vorteil: Einfach zu implementieren und zu testen.
  - Nachteil: Hohe Interrupt-Latenzzeiten für gleiche Prioritäten.



Beispielcode EXTI15_10_IRQHandler

```
/**
*  @brief   This function handles EXTI line[15:10] interrupts.
*           The button is located at line 11.
*           If button is pressed, the pending bit is reset.
*/
void EXTI15_10_IRQHandler(void)
{
    /* Check if line 11 is active */
    if((EXTI->PR1 & (1 << 11)) != 0){
        /* clear pending bit by writing a 1 to its position */
        EXTI->PR1 = (1 << 11);
    }
}
```