Interrupt (Grundlagen)

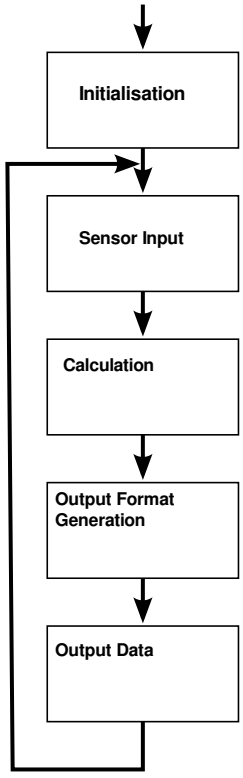
ARM Cortex M4 - TM4C1294

Vorlesung Mikroprozessortechnik

HAW Hamburg

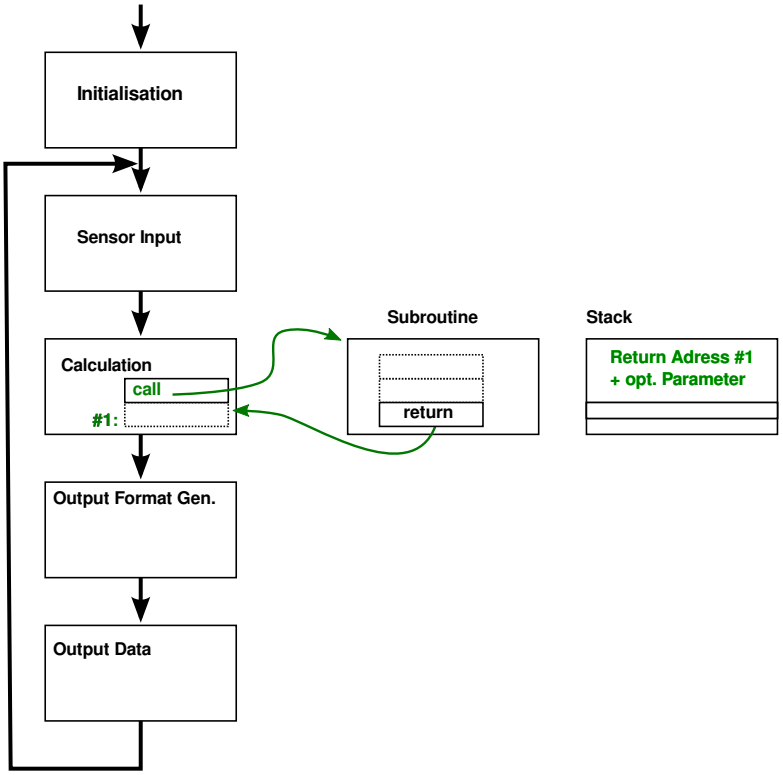
12. Mai 2018

Einfache Steuerschleife (Beispiel)



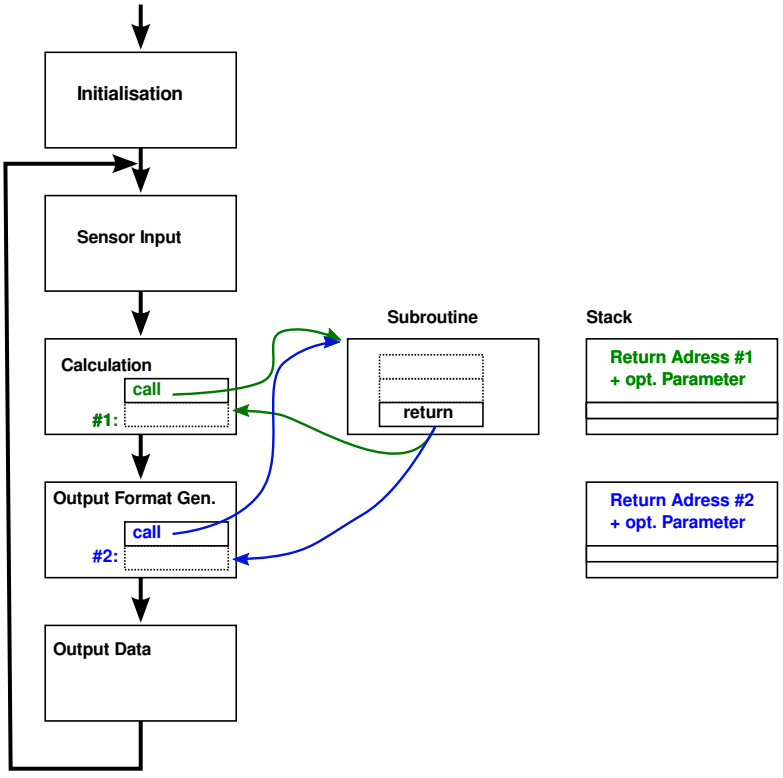
Eine einfache Programmstruktur hat eine sequentielle Folge der Anweisungen. Die Befehle haben nacheinanderfolgen Addressen. Im Beispiel gibt es nur eine Ausnahme, welche als Sprung zum Anfang der wiederholten Sequenz die Schleife bildet.

Steuerschleife und eine Subroutine



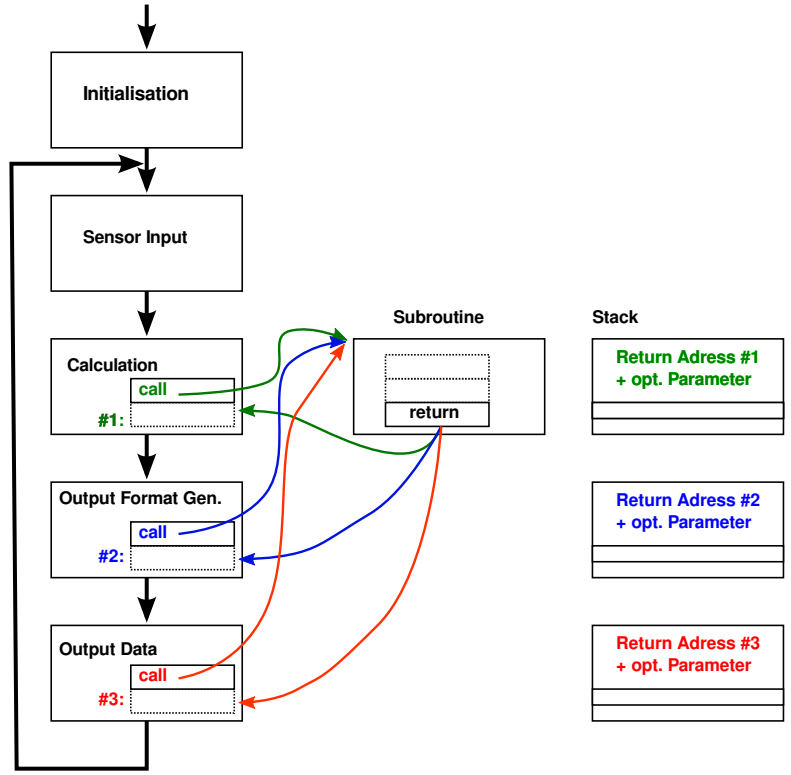
Der Funktionsaufruf verlässt als Subroutine die Sequenz der Befehle. Der Kontext, wie u.a. die nächste Adresse der Sequenz, muß zwischengespeichert werden. Das Zwischenspeichern erfolgt auf dem Stackbereich des Speichers.

Steuerschleife und zwei Subroutinen



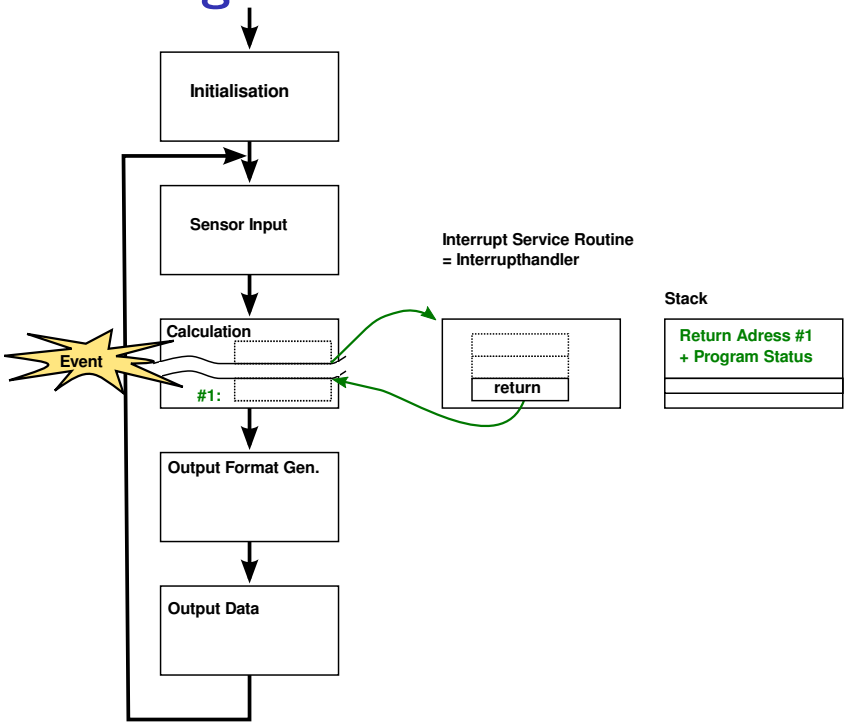
Subroutinen werden durch einen Call-Befehl aufgerufen. Sie erfolgen also unter der Steuerung des Programms.

Steuerschleife und drei Subroutinen



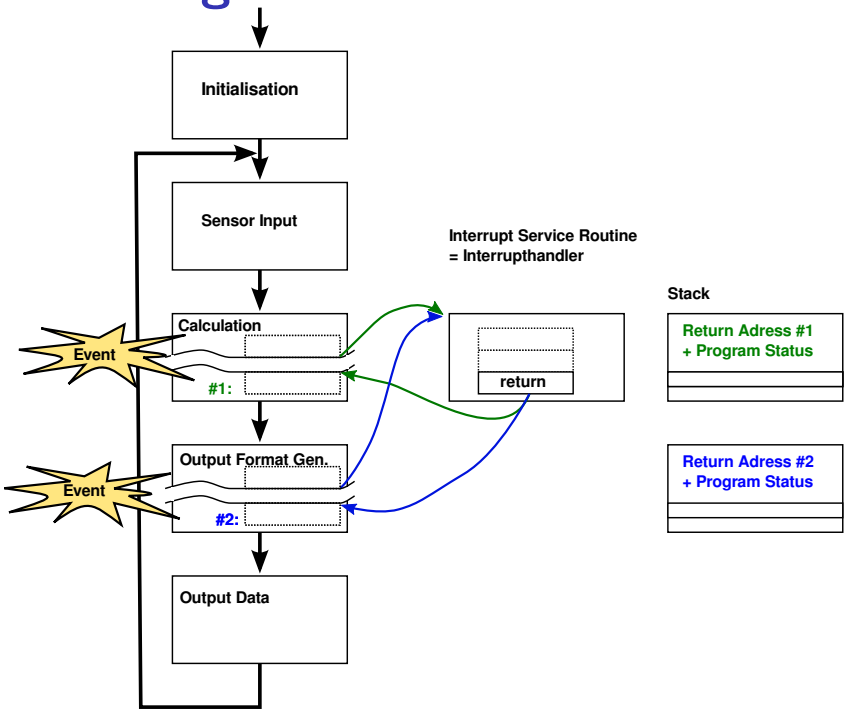
Subroutine können vielfach benutzt werden. Sie können bei jedem Aufruf andere Parameter erhalten und unterschiedliche Rückgaben liefern. Die Aufrufparameter und Rückgaben werden von Hochsprachen unterstützt.

Steuerschleife wird durch ein Events und die Event-Behandlung unterbrochen



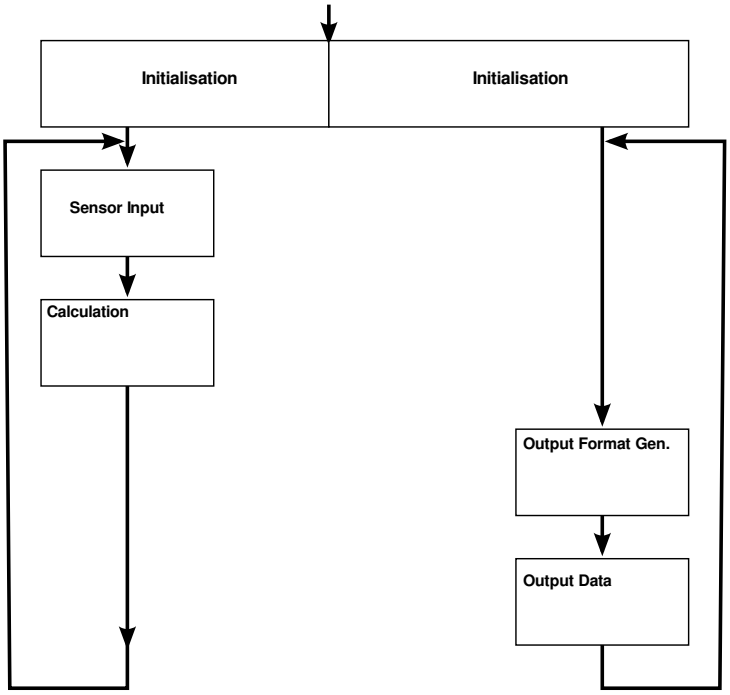
Eine Service Routine wird nicht durch den Programmlauf gesteuert. Sie hat keinen Aufruf-Befehl (Call). Ein zum Programm asynchrones Ereignis löst die Behandlung aus.

Steuerschleife wird durch mehrfache Events und die Event-Behandlungen unterbrochen



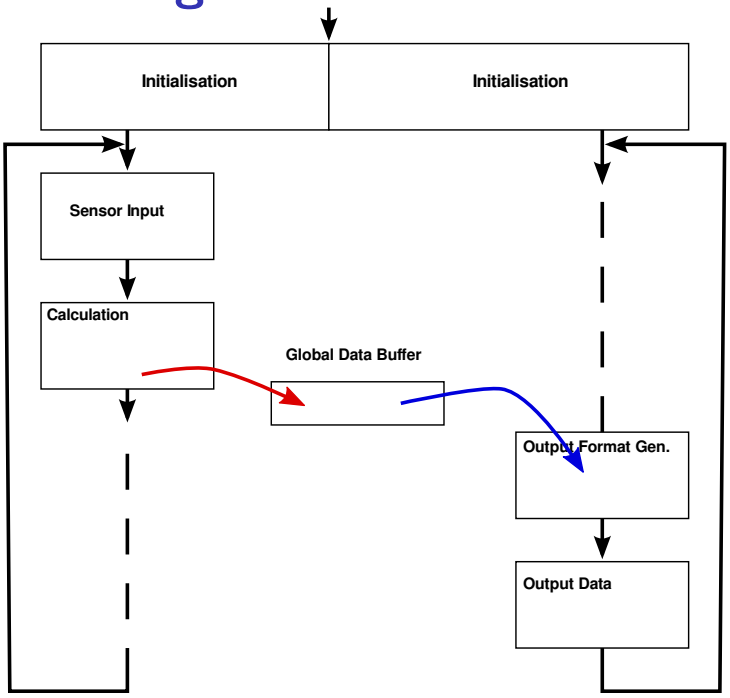
Events können mehrfach auftreten und werden durch Service Routinen behandelt. Um den Kontext zwischenzuspeichern wird ebenfalls der Stack-Bereich im Speicher benutzt.

Zwei funktionell notwendige 'Schleifen' mit asynchronem Durchlauf



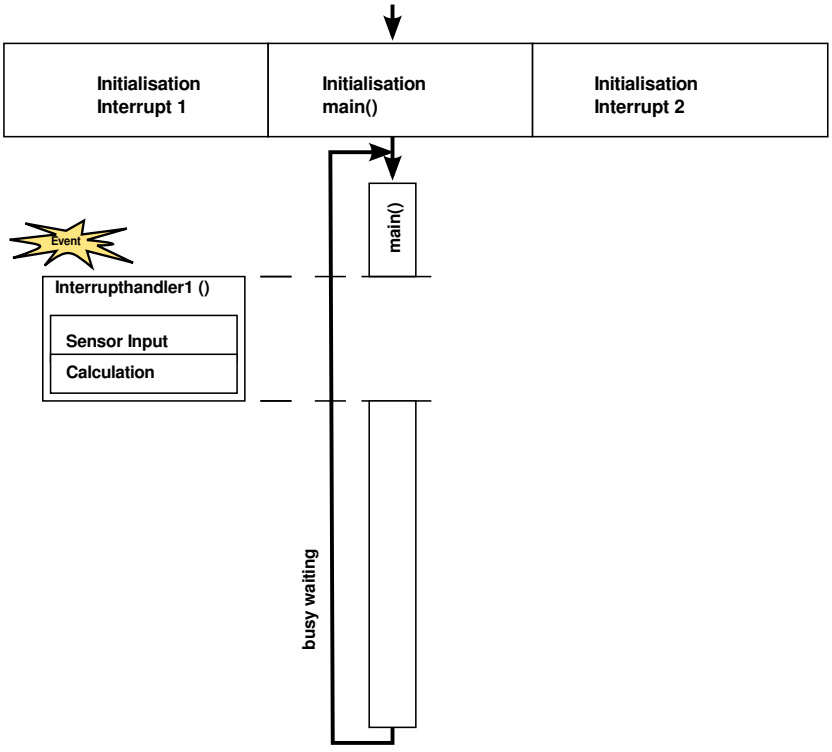
Haben die Input-Funktion und die Output-Funktion ein unterschiedliches Zeitverhalten, so werden getrennte Schleifen erforderlich. Diese sind aber nicht einfach mit sequentiellen Anweisungen programmierbar.

Asynchron laufende 'Schleifen' erfordern eine globale Datenübergabe

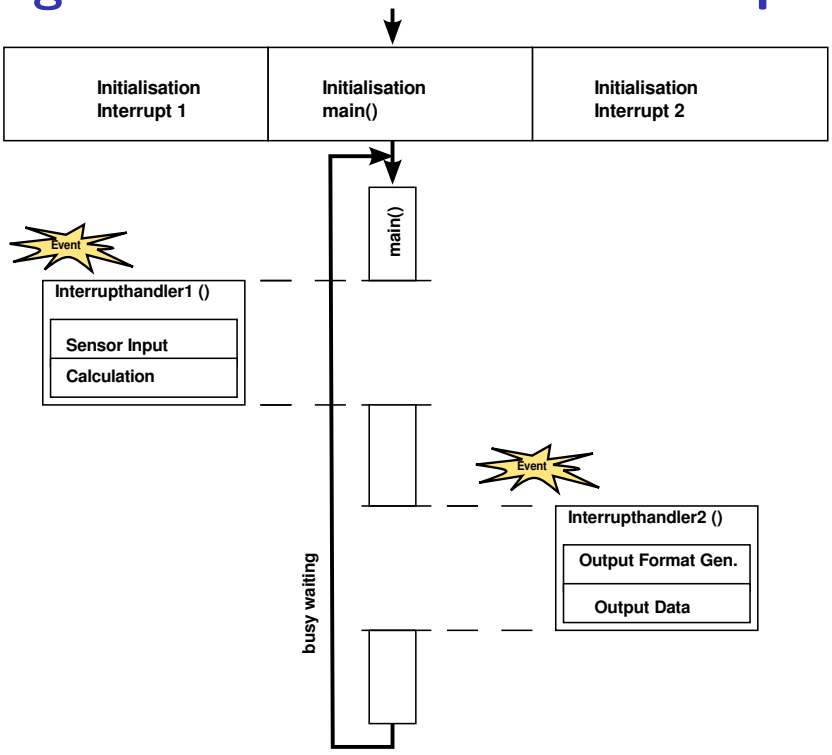


Die Nutzung von Aufruf-Parameter und Rückgabewerten - wie bei Subroutinen/Funktionen - ist grundsätzlich nicht möglich. Es muß auf globale Variablen zurückgegriffen werden.

Verlagerung der Funktion in die Interrupthandler I

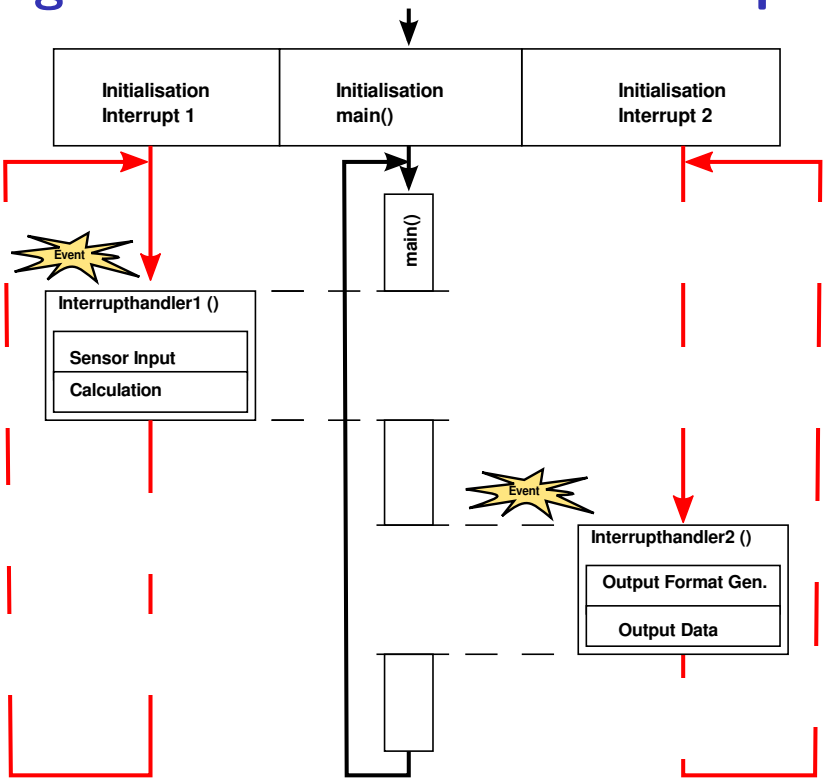


Verlagerung der Funktion in die Interrupthandler II



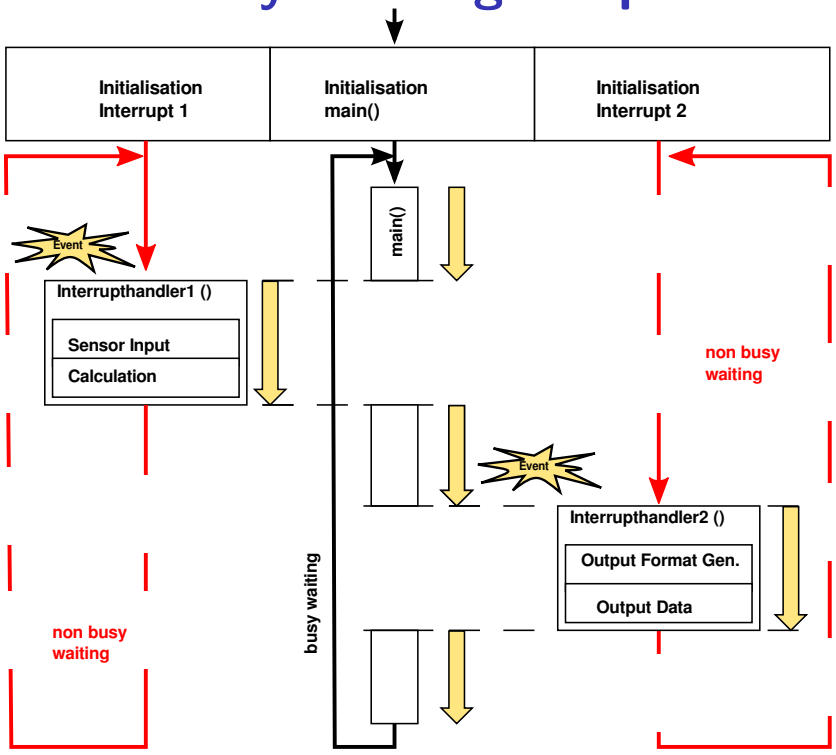
Die Interrupt-Service-Routinen sind reaktiv. Sie werden ausgelöst von einem Ereignis ausserhalb des Programmlaufs (Event Trigger).

Verlagerung der Funktion in die Interrupthandler III



Die funktional notwendigen Schleifen werden durch die (wiederholten) Interrupt-Service-Routinen gebildet. Die Prozessorresourcen werden nach dem Ablauf wieder freigeben.

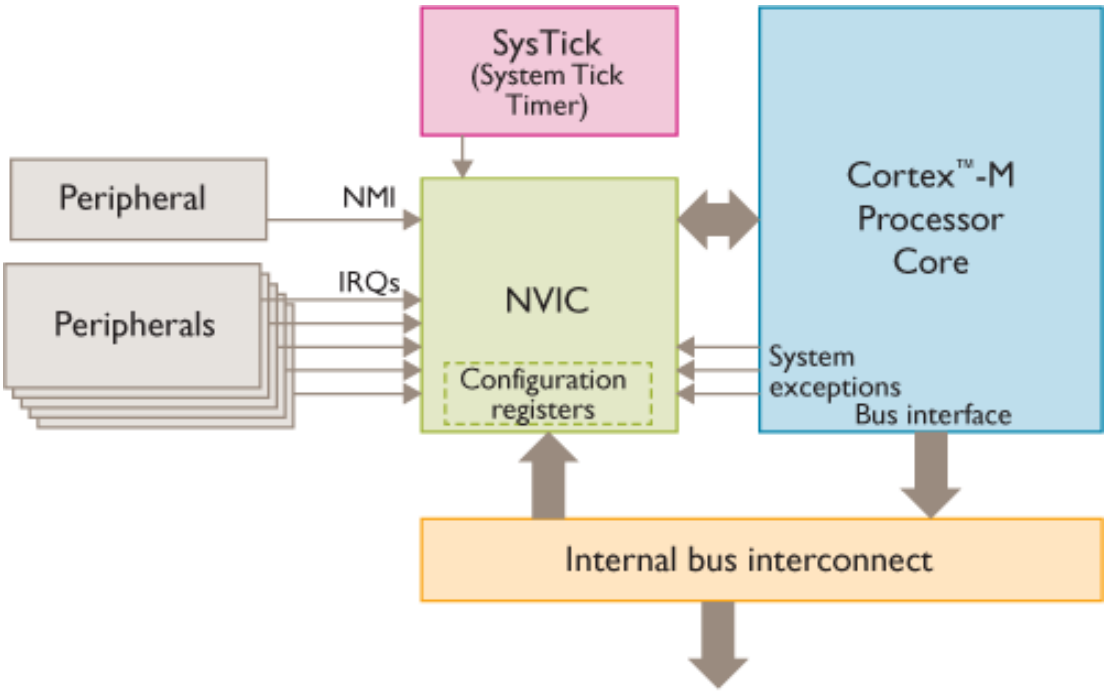
Busy- und Non-Busy-Waiting-Loops



Nur die Busy-Waiting-Loop ist eine Warteschleife im Programmcode.

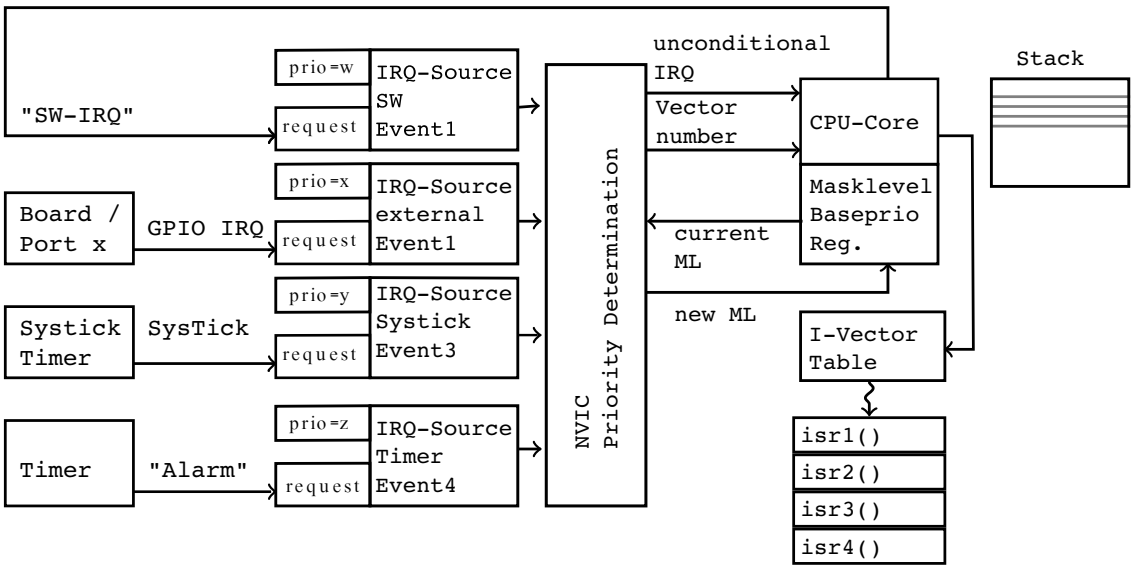
Die Non-Busy-Waiting-Loops erfüllen eine vergleichbare Funktion ohne eine Warteschleife im Programmcode zu erfordern.

Nested Vector Interrupt Controller NVIC



Source: ARM Cortex-M Technologies, www.arm.com

Interrupt Source→Request→Priority-Determination→Acceptance→Handling→Return



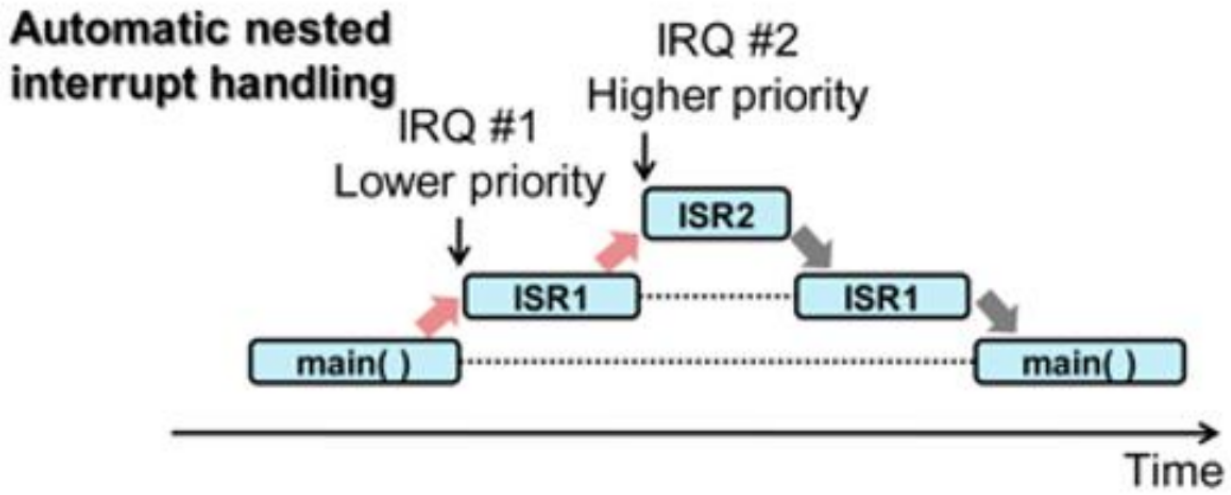
Source inspired and modified
D. Lohmann,SLOTH "Threads as Interrupts"
Univ. Erlangen

Zustände der Interrupts

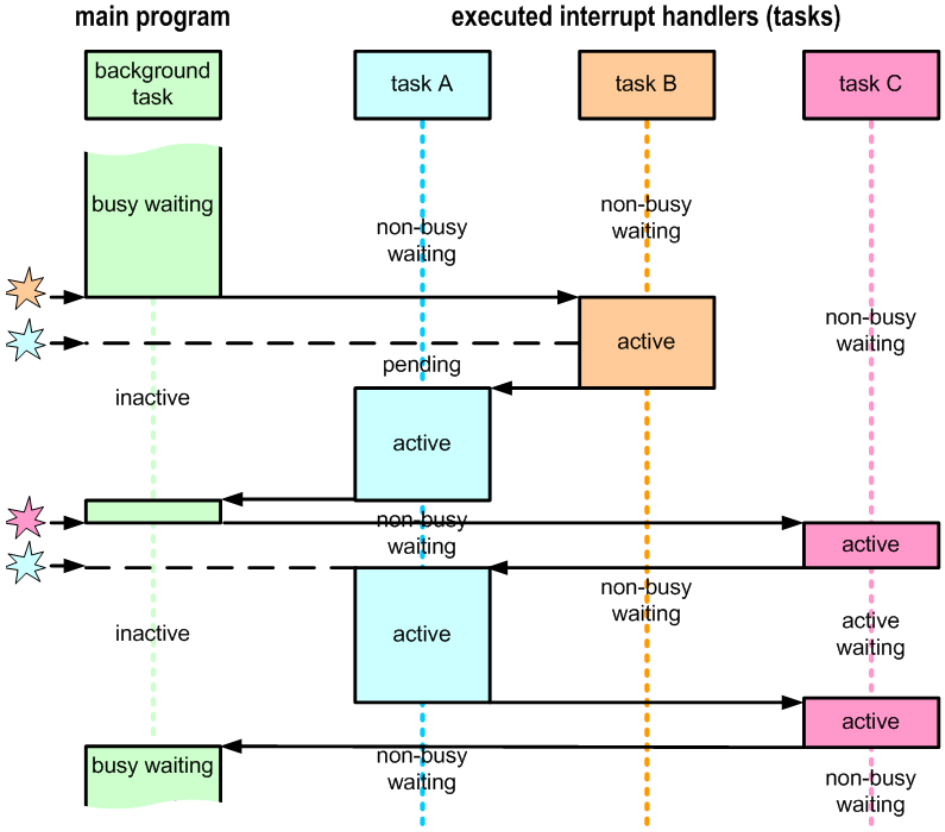
- Inactive The Interrupt/Exception is not active and not pending.
- Pending The Interrupt/Exception is waiting to be serviced by the processor. An interrupt request from a peripheral or from software can change the state of the corresponding interrupt to pending.
- Active. An Interrupt/Exception that is being serviced by the processor but has not completed.

  Note: An exception handler can interrupt the execution of another exception handler. In this case, both exceptions are in the active state.
- Active and Pending. The Interrupt/Exception is being serviced by the processor, and there is a pending exception from the same source.

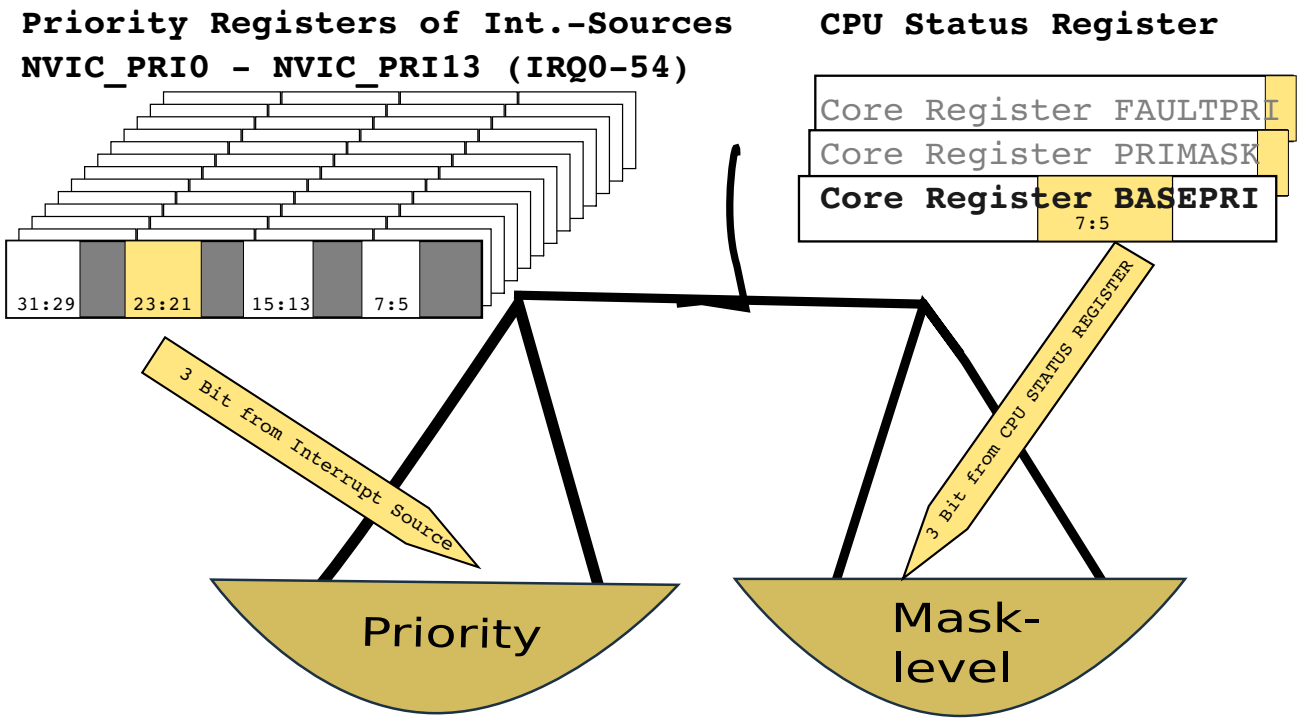
Preemption, Nesting, Prioriätslevel



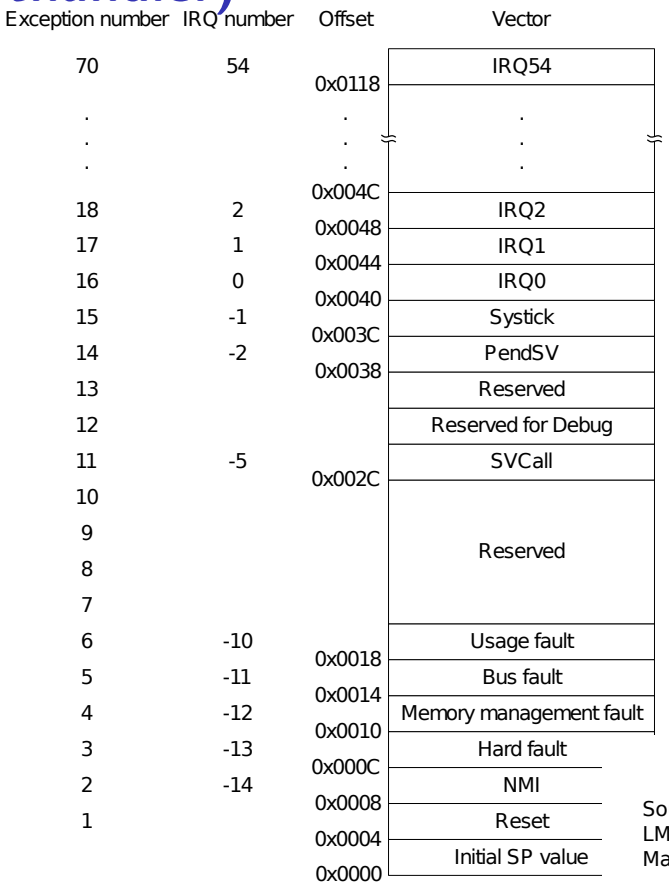
Preemption, Nesting, Prioriätslevel



Priority Determination



Interrupt Vector Table (Anfangsaddressen der Interrupthandler)



Source: Texas Instruments
LM3S9B92-Data-Sheet.pdf
March 19, 2011

GPIO Interrupt Sense Register

GPIO_PORTx_AHB_IS_R, GPIO_PORTy_IS_R, x=A-J, y=K-N,P-Q

GPIO Interrupt Sense (GPIOIS)

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
reserved															
Type	RO	RO	RO	RO	RO	RO	RO	RO	RO	RO	RO	RO	RO	RO	RO	RO
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
reserved								IS							
Type	RO	RO	RO	RO	RO	RO	RO	RO	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Bit/Field	Name	Type	Reset	Description
7:0	IS	R/W	0x00	GPIO Interrupt Sense

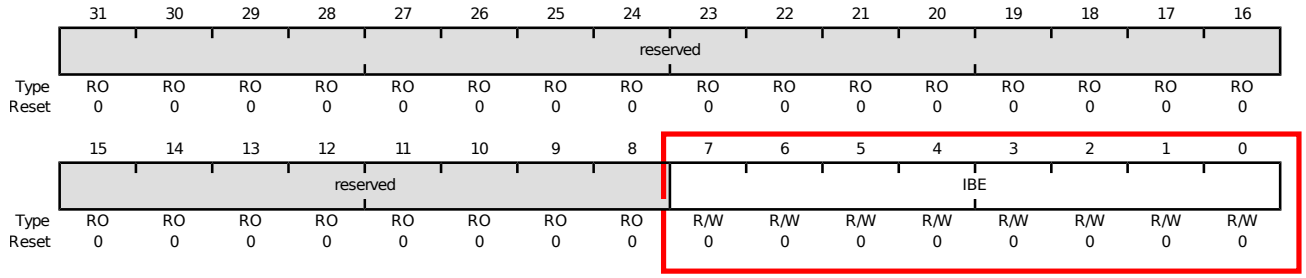
Value	Description
0	The edge on the corresponding pin is detected (edge-sensitive).
1	The level on the corresponding pin is detected (level-sensitive).

Source Datasheet p.761 TM4C1294NCPDT Microcontroller Texas Instruments June 18, 2014

GPIO Interrupt Both Edges Register

GPIO_PORTx_AHB_IBE_R, GPIO_PORTy_IBE_R, x=A–J, y=K–N, P–Q

GPIO Interrupt Both Edges (GPIOIBE)



Bit/Field	Name	Type	Reset	Description
7:0	IBE	R/W	0x00	GPIO Interrupt Both Edges

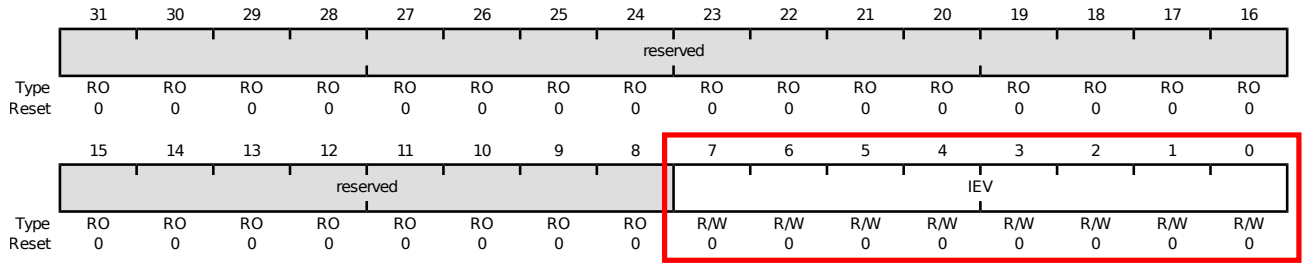
Value	Description
0	Interrupt generation is controlled by the **GPIO Interrupt Event (GPIOIEV)** register (see page 415).
1	Both edges on the corresponding pin trigger an interrupt.

Source Datasheet p.762 TM4C1294NCPDT Microcontroller Texas Instruments June 18, 2014

GPIO Interrupt Event Register

GPIO_PORTx_AHB_IEV_R, GPIO_PORTy_IEV_R, x=A–J, y=K–N, P–Q

GPIO Interrupt Event (GPIOIEV)



Bit/Field	Name	Type	Reset	Description
7:0	IEV	R/W	0x00	GPIO Interrupt Event

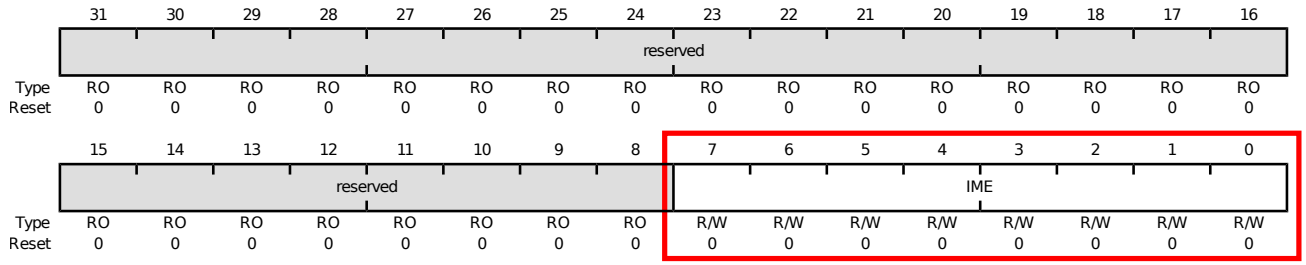
Value	Description
	A falling edge or a Low level on the corresponding pin triggers an interrupt.
	A rising edge or a High level on the corresponding pin triggers an interrupt.

Source Datasheet p.763 TM4C1294NCPDT Microcontroller Texas Instruments June 18, 2014

GPIO Interrupt Mask Register

GPIO_PORTx_AHB_IM_R, GPIO_PORTy_IM_R, x=A-J, y=K-N, P-Q

GPIO Interrupt Mask (GPIOIM)



Bit/Field	Name	Type	Reset	Description
7:0	IME	R/W	0x00	GPIO Interrupt Mask Enable

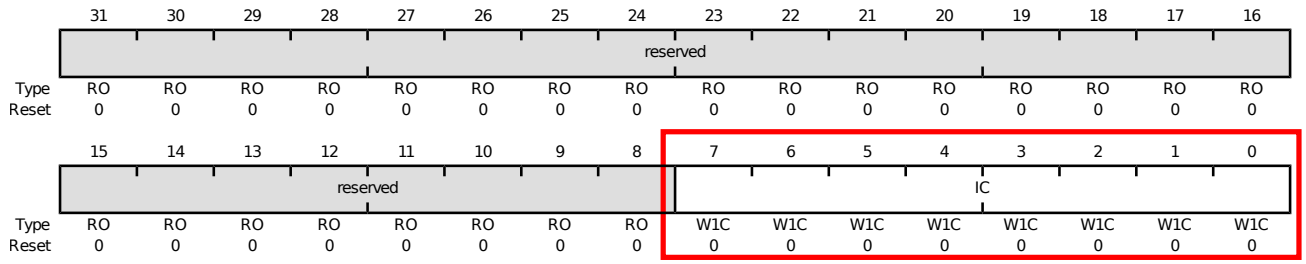
Value	Description
0	The interrupt from the corresponding pin is masked.
	The interrupt from the corresponding pin is sent to the interrupt controller.

Source Datasheet p.764 TM4C1294NCPDT Microcontroller Texas Instruments June 18, 2014

GPIO Interrupt Clear Register

GPIO_PORTx_AHB_ICR_R, GPIO_PORTy_ICR_R, x=A-J, y=K-N, P-Q

GPIO Interrupt Clear (GPIOICR)



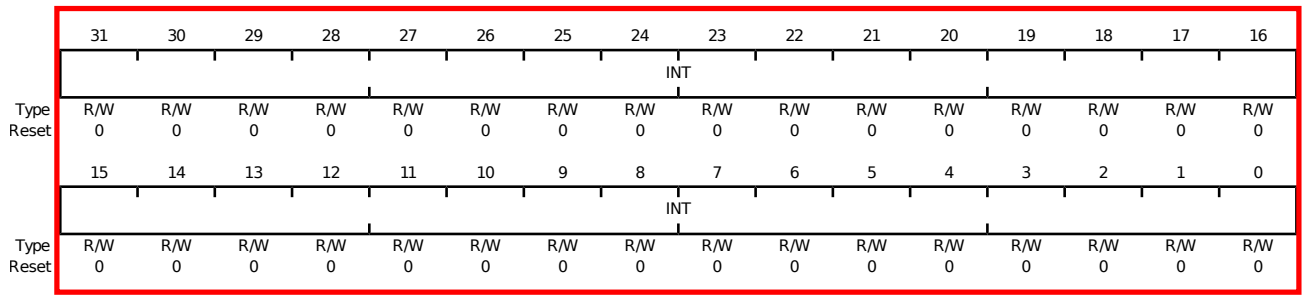
Bit/Field	Name	Type	Reset	Description
7:0	IC	W1C	0x00	GPIO Interrupt Clear

Value	Description
1	The corresponding interrupt is cleared.
0	The corresponding interrupt is unaffected.

Source Datasheet p.769 TM4C1294NCPDT Microcontroller Texas Instruments June 18, 2014

NVIC Interrupt Set Enable Register NVIC_ENx_R, x=0-3

Interrupt 0-31 Set Enable (EN0)
Interrupt 32-54 Set Enable (EN1) *)



Bit/Field	Name	Type	Reset	Description
31:0	INT	R/W	0x0000.0000	Interrupt Enable

*) 22:0

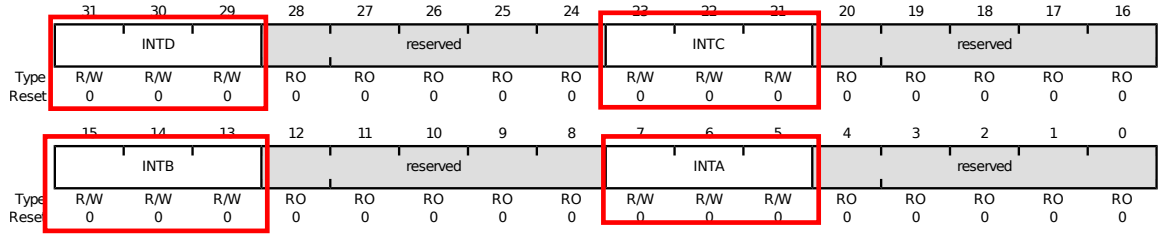
Value	Description
0	On a read, indicates the interrupt is disabled. On a write, no effect.
1	On a read, indicates the interrupt is enabled. On a write, enables the interrupt.

A bit can only be cleared by setting the corresponding `INT[n]` bit in the **DISn** register.

Source Datasheet p.153ff TM4C1294NCPDT Microcontroller Texas Instruments June 18, 2014

NVIC Priority Register NVIC_PIOx_R, x=0-28

Interrupt 0-3 Priority (PRI0) Interrupt 4-7 Priority (PRI1)
Interrupt 8-11 Priority (PRI2) Interrupt 12-15 Priority (PRI3)
Interrupt 16-19 Priority (PRI4) Interrupt 20-23 Priority (PRI5)
Interrupt 24-27 Priority (PRI6) Interrupt 28-31 Priority (PRI7)
Interrupt 32-35 Priority (PRI8) Interrupt 36-39 Priority (PRI9)
Interrupt 40-43 Priority (PRI10) Interrupt 44-47 Priority (PRI11)
Interrupt 48-51 Priority (PRI12) Interrupt 52-54 Priority (PRI13)



Bit/Field	Name	Type	Reset	Description
31:29 or 23:21 or 15:13 or 17:5	INTD INTC INTB INTA	R/W R/W R/W R/W	0x0 0x0 0x0 0x0	This field holds a value, 0-7, for the interrupt priority with the number [4 n +3] or [4 n +2] or [4 n +1] or [4 n] n is the number of the interrupt priority register (n=0 for PRI0, and so on). The lower the value, the greater the priority of the corresponding interrupt.

Source Datasheet p.159ff TM4C1294NCPDT Microcontroller Texas Instruments June 18, 2014

Eintrag des Handlers in die Interrupt Vector Table (IVT) → Code im EMIL!

↪ Direkt im Startup-Code:

Der Startup-Code wird vor der main() ausgeführt

Der Startup-Code enthält Initialisierungen von Daten/Codesegmenten, die im Flash-ROM gespeichert werden (Release-Compilation)

- Praktischer Ansatz : Modifikation von des Files startup.ccs.c

1: Kopieren des Files startup.ccs.c aus der Stellarisware ins Project

2: Eintrag eines Funktionsprototypen des Handlers (external declaration) dort

3: Eintrag des EIGENEN Handlernames in die Datenstruktur der IVT anstelle des Defaulteintrages für die passende(n) Quelle(n)

↪ Durch Registrieren in einer Shadow-IVT:

- Bibliotheksfunktion der Firmware (Tivaware)
  `IntRegister( <Source Nr.>, <Handler Name>)`

  Die Funktion kopiert die IVT in RAM, setzt das NVIC-Offset-Register auf die kopierte IVT und trägt die Anfangsaddresse des Handlers(= 'Handler Name') an die Stelle des Eintrags für die 'Source Nr.' ein