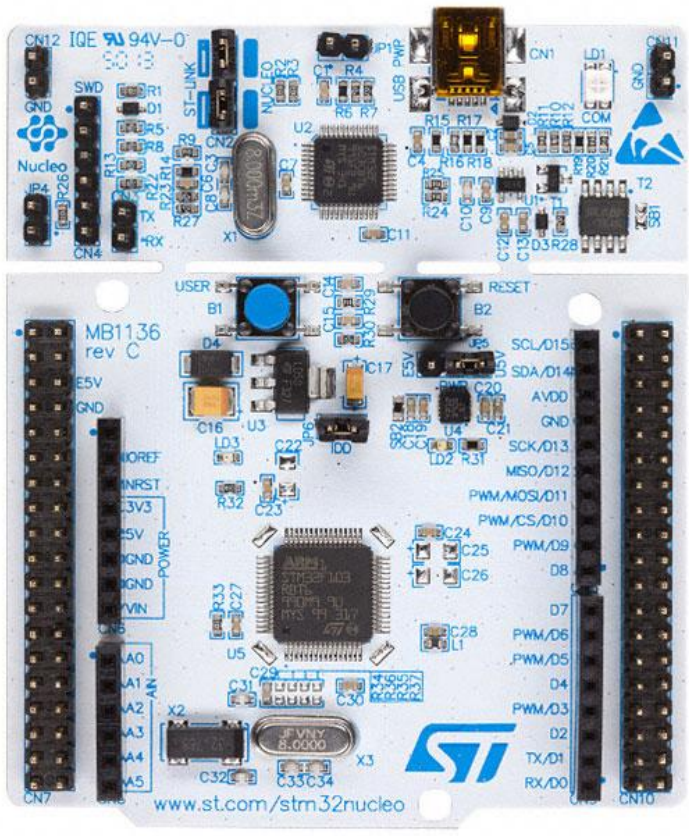
GHSE Gewerbliche und Hauswirtschaftlich-Sozialpflegerische Schulen Emmendingen
unesco-projekt-schulen

Mikrocontroller-Labor

Formelsammlung



Nucleo STM32F411RE

Stand: 21.01.2025

Inhaltsverzeichnis

1. Pinout Nucleo STM32F411

Labels usable in code

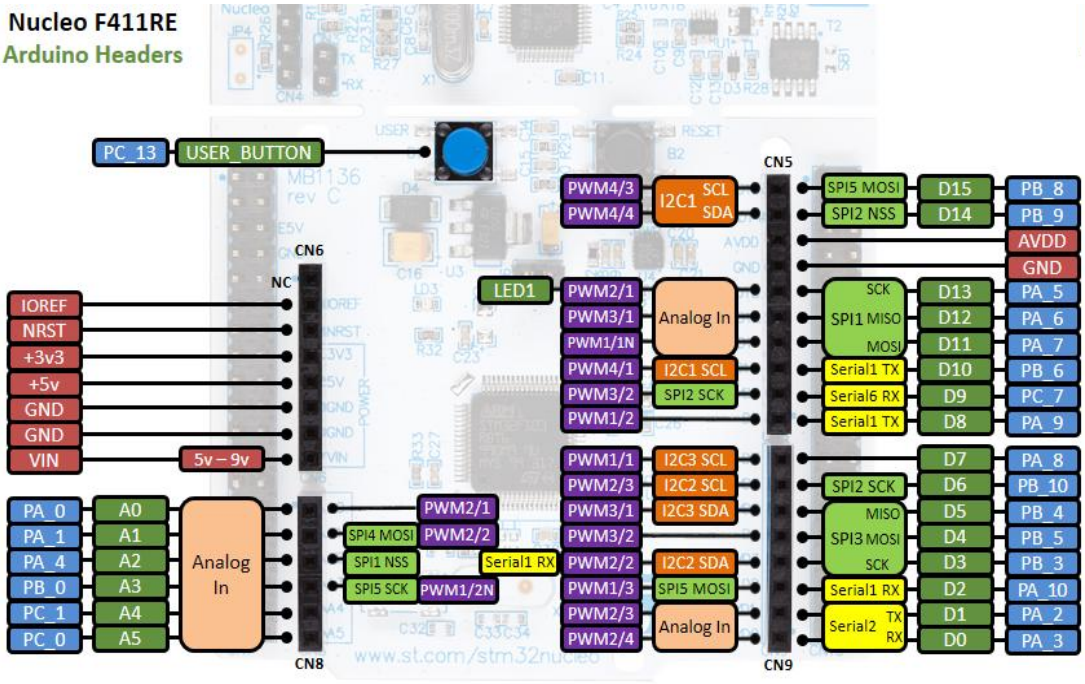
PX_Y	MCU pin without conflict
PX_Y	MCU pin connected to other components See **PeripheralPins.c** (link below) for more information
XXX	Arduino connector names (A0, D1, ...)
XXX	LEDs and Buttons (LED_1, USER_BUTTON, ...)

Labels not usable in code (for information only)

XXX	Serial pins (USART/UART)
XXX	SPI pins
XXX	I2C pins
XXX	PWMOut pins (TIMER n/c[N]) n = Timer number c = Channel N = Inverted channel
XXX	AnalogIn (ADC) and AnalogOut pins (DAC)
XXX	CAN pins
XXX	Power and control pins (3V3, GND, RESET, ...)

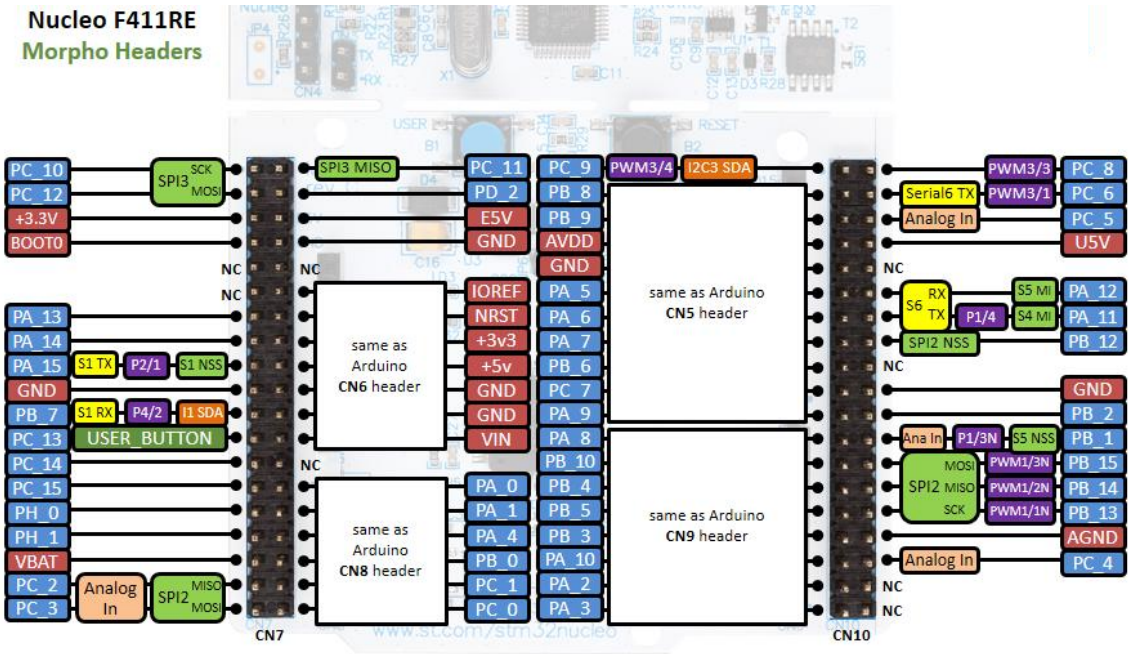
Pinzuordnung Arduino Header

Nucleo F411RE
Arduino Headers

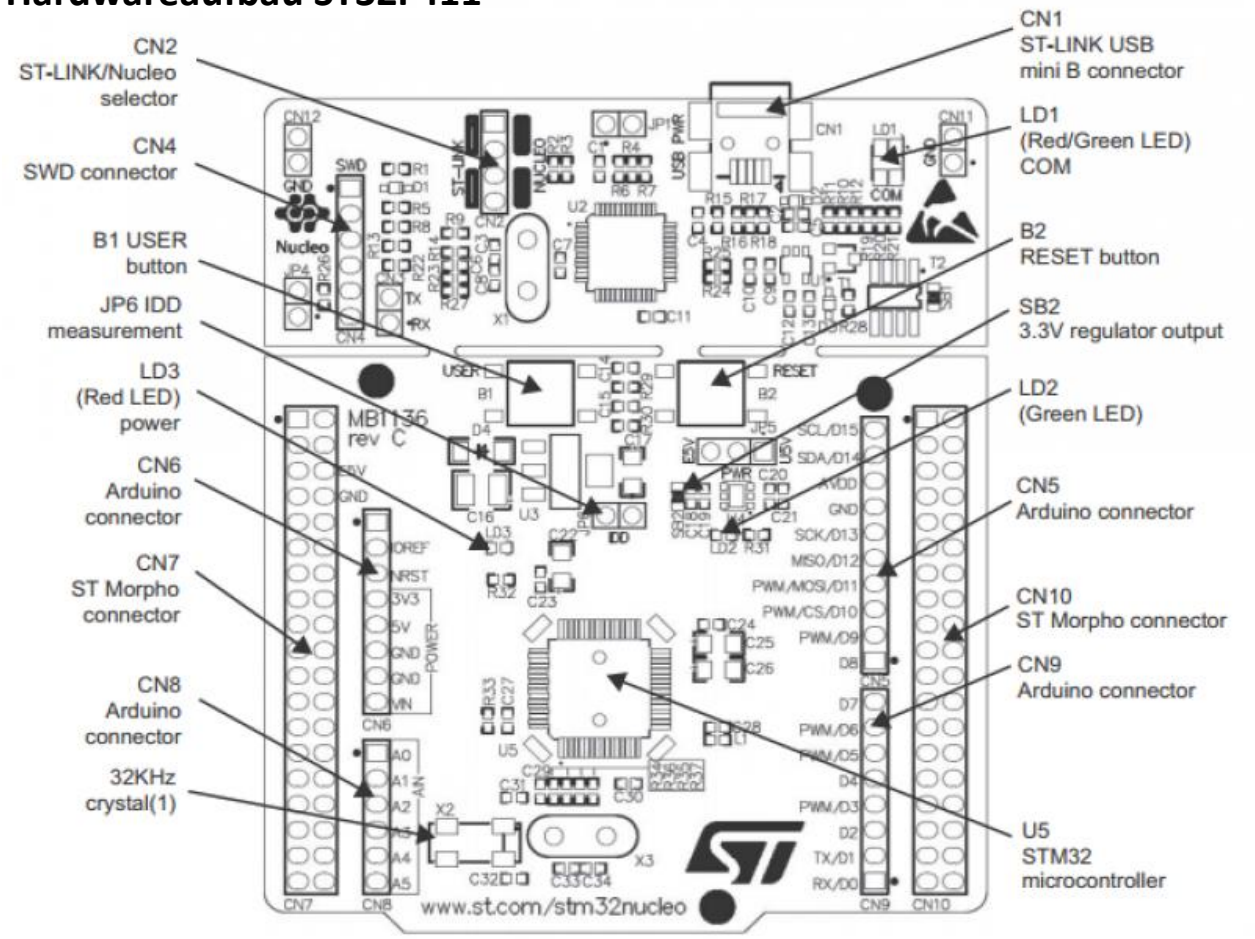


Pinzuordnung Morpho Header

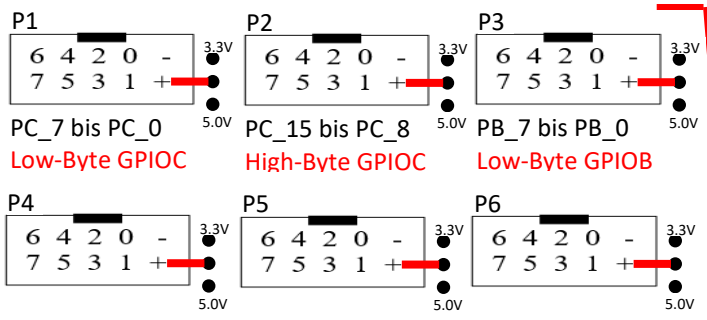
Nucleo F411RE
Morpho Headers



2. Hardwareaufbau ST32F411

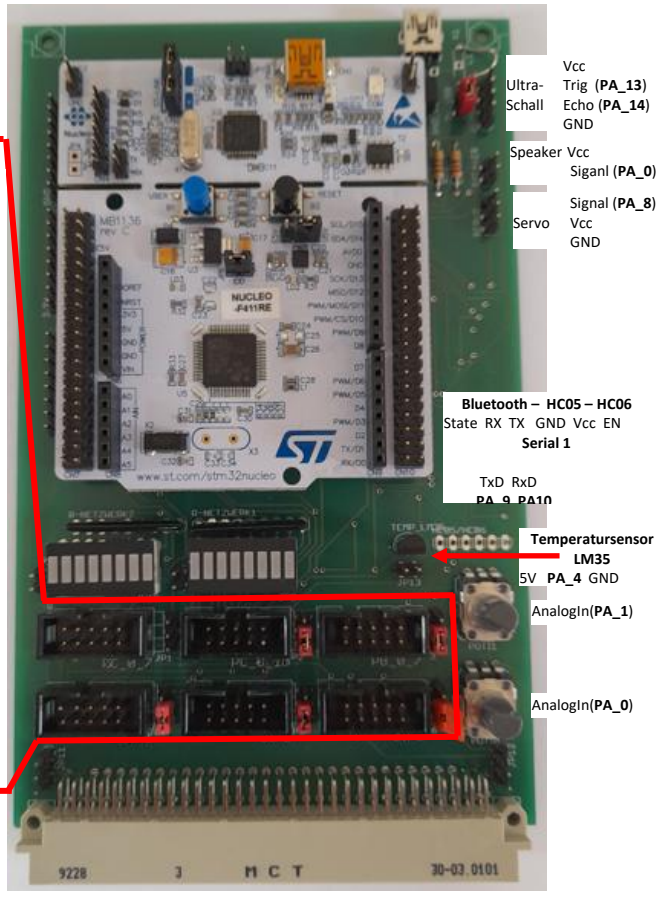


Belegung der ML10 Buchsen:

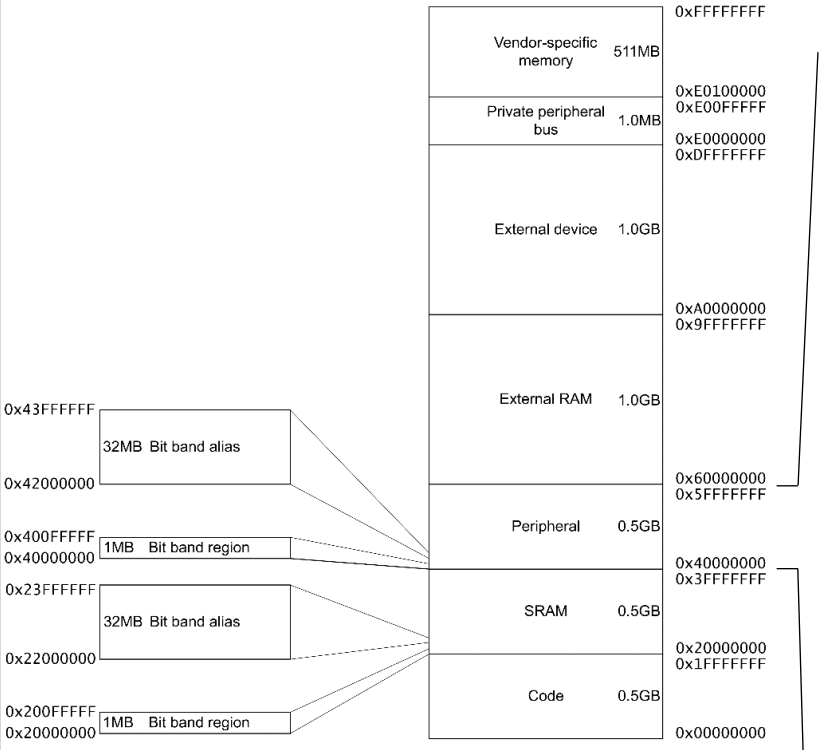


Pin	P4		P5		P6	
.0	PA_10	RxD	PB_7	RxD	PC_6	TxD
.1	PA_9	TxD	PB_6	TxD	PC_7	RxD
.2	PB_9	SDA	PB_9	SDA	PB_9	SDA
.3	PB_8	SCL	PB_8	SCL	PB_8	SCL
.4	PA_15	SS	PA_15	SS	PB_12	SS
.5	PA_6	MISO	PA_6	MISO	PB_14	MISO
.6	PA_5	SCK	PA_5	SCK	PB_13	SCK
.7	PA_7	MOSI	PA_7	MOSI	PB_15	MOSI

UART **I2C** **SPI**



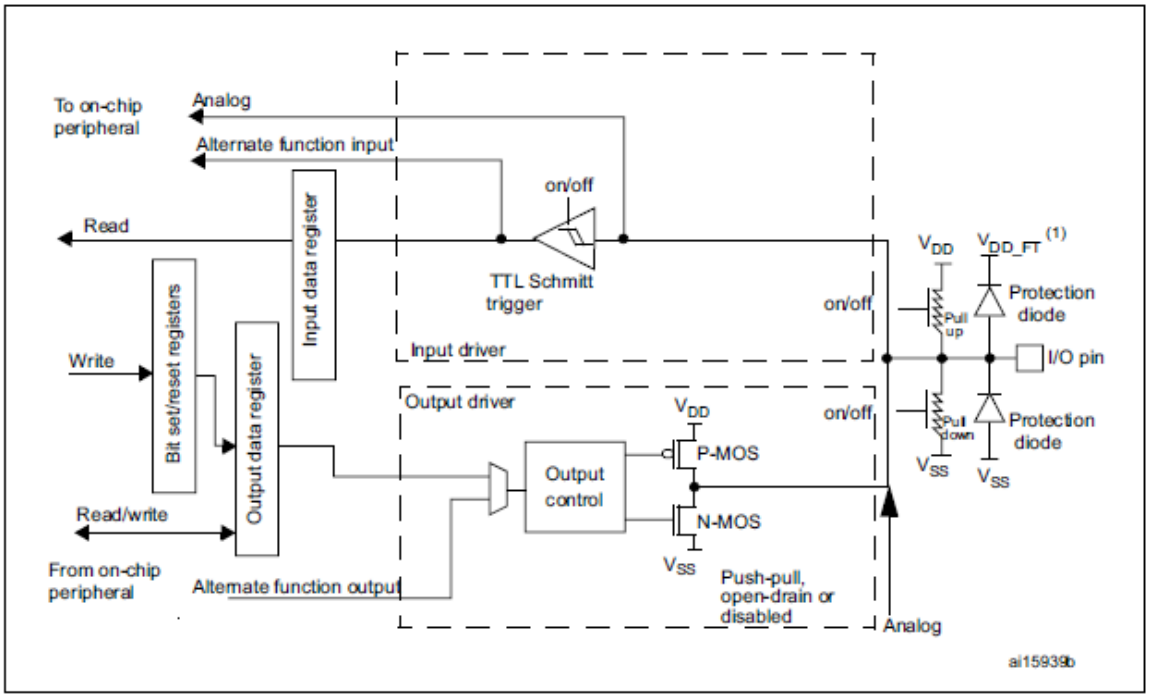
2.1. Memory Map



Boundary address	Peripheral	Bus
0x5000 0000 - 0x5003 FFFF	USB OTG FS	AHB2
0x4002 6400 - 0x4002 67FF	DMA2	AHB1
0x4002 6000 - 0x4002 63FF	DMA1	
0x4002 3C00 - 0x4002 3FFF	Flash interface register	
0x4002 3800 - 0x4002 3BFF	RCC	
0x4002 3000 - 0x4002 33FF	CRC	
0x4002 1C00 - 0x4002 1FFF	GPIOH	
0x4002 1000 - 0x4002 13FF	GPIOE	
0x4002 0C00 - 0x4002 0FFF	GPIOD	
0x4002 0800 - 0x4002 0BFF	GPIOC	
0x4002 0400 - 0x4002 07FF	GPIOB	
0x4002 0000 - 0x4002 03FF	GPIOA	

Boundary address	Peripheral	Bus
0x4001 5000 - 0x4001 53FFF	SPI5/I2S5	APB2
0x4001 4800 - 0x4001 4BFF	TIM11	
0x4001 4400 - 0x4001 47FF	TIM10	
0x4001 4000 - 0x4001 43FF	TIM9	
0x4001 3C00 - 0x4001 3FFF	EXTI	
0x4001 3800 - 0x4001 3BFF	SYSCFG	
0x4001 3400 - 0x4001 37FF	SPI4/I2S4	
0x4001 3000 - 0x4001 33FF	SPI1/I2S1	
0x4001 2C00 - 0x4001 2FFF	SDIO	
0x4001 2000 - 0x4001 23FF	ADC1	
0x4001 1400 - 0x4001 17FF	USART6	
0x4001 1000 - 0x4001 13FF	USART1	
0x4001 0000 - 0x4001 03FF	TIM1	
0x4000 7000 - 0x4000 73FF	PWR	APB1
0x4000 5C00 - 0x4000 5FFF	I2C3	
0x4000 5800 - 0x4000 5BFF	I2C2	
0x4000 5400 - 0x4000 57FF	I2C1	
0x4000 4400 - 0x4000 47FF	USART2	
0x4000 4000 - 0x4000 43FF	I2S3ext	
0x4000 3C00 - 0x4000 3FFF	SPI3 / I2S3	
0x4000 3800 - 0x4000 3BFF	SPI2 / I2S2	
0x4000 3400 - 0x4000 37FF	I2S2ext	
0x4000 3000 - 0x4000 33FF	IWDG	
0x4000 2C00 - 0x4000 2FFF	WWDG	
0x4000 2800 - 0x4000 2BFF	RTC & BKP Registers	
0x4000 0C00 - 0x4000 0FFF	TIM5	
0x4000 0800 - 0x4000 0BFF	TIM4	
0x4000 0400 - 0x4000 07FF	TIM3	
0x4000 0000 - 0x4000 03FF	TIM2	

2.2. GPIO Hardwareaufbau und Register



RCC AHB1 peripheral clock enable register (RCC_AHB1ENR)

Address offset: 0x30

Reset value: 0x0000 0000

Access: no wait state, word, half-word and byte access.

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Reserved									DMA2EN	DMA1EN	Reserved				
									rw	rw					

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reserved			CRCEN	Reserved				GPIOH EN	Reserved		GPIOEEN	GPIOD EN	GPIOC EN	GPIOB EN	GPIOA EN
			rw					rw			rw	rw	rw	rw	rw

Bit 1 **GPIOBEN:** IO port B clock enable
Set and cleared by software.
0: IO port B clock disabled
1: IO port B clock enabled

Bit 0 **GPIOAEN:** IO port A clock enable
Set and cleared by software.
0: IO port A clock disabled
1: IO port A clock enabled

GPIO port mode register (GPIOx_MODER) (x = A..E and H)

Address offset: 0x00

Reset values:

- 0xA800 0000 for port A
- 0x0000 0280 for port B
- 0x0000 0000 for other ports

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
MODER15[1:0]		MODER14[1:0]		MODER13[1:0]		MODER12[1:0]		MODER11[1:0]		MODER10[1:0]		MODER9[1:0]		MODER8[1:0]	
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
MODER7[1:0]		MODER6[1:0]		MODER5[1:0]		MODER4[1:0]		MODER3[1:0]		MODER2[1:0]		MODER1[1:0]		MODER0[1:0]	
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Bits 2y:2y+1 **MODERy[1:0]:** Port x configuration bits (y = 0..15)
These bits are written by software to configure the I/O direction mode.
00: Input (reset state)
01: General purpose output mode
10: Alternate function mode
11: Analog mode

GPIO port output type register (GPIOx_OTYPER) (x = A..E and H)

Address offset: 0x04

Reset value: 0x0000 0000

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Reserved															
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
OT15	OT14	OT13	OT12	OT11	OT10	OT9	OT8	OT7	OT6	OT5	OT4	OT3	OT2	OT1	OT0
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Bits 31:16 Reserved, must be kept at reset value.

Bits 15:0 **OTy**: Port x configuration bits (y = 0..15)
These bits are written by software to configure the output type of the I/O port.
0: Output push-pull (reset state)
1: Output open-drain

GPIO port output speed register (GPIOx_OSPEEDR) (x = A..E and H)

Address offset: 0x08

Reset values:

- 0x0C00 0000 for port A
- 0x0000 00C0 for port B
- 0x0000 0000 for other ports

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
OSPEEDR15 [1:0]		OSPEEDR14 [1:0]		OSPEEDR13 [1:0]		OSPEEDR12 [1:0]		OSPEEDR11 [1:0]		OSPEEDR10 [1:0]		OSPEEDR9 [1:0]		OSPEEDR8 [1:0]	
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
OSPEEDR7[1:0]		OSPEEDR6[1:0]		OSPEEDR5[1:0]		OSPEEDR4[1:0]		OSPEEDR3[1:0]		OSPEEDR2[1:0]		OSPEEDR1 [1:0]		OSPEEDR0 1:0]	
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Bits 2y:2y+1 **OSPEEDRy[1:0]:** Port x configuration bits (y = 0..15)
These bits are written by software to configure the I/O output speed.
00: Low speed
01: Medium speed
10: Fast speed
11: High speed

Note: Refer to the product datasheets for the values of OSPEEDRy bits versus V_{DD} range and external load.

GPIO port pull-up/pull-down register (GPIOx_PUPDR) (x = A..E and H)

Address offset: 0x0C

Reset values:

- 0x6400 0000 for port A
- 0x0000 0100 for port B
- 0x0000 0000 for other ports

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
PUPDR15[1:0]		PUPDR14[1:0]		PUPDR13[1:0]		PUPDR12[1:0]		PUPDR11[1:0]		PUPDR10[1:0]		PUPDR9[1:0]		PUPDR8[1:0]	
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
PUPDR7[1:0]		PUPDR6[1:0]		PUPDR5[1:0]		PUPDR4[1:0]		PUPDR3[1:0]		PUPDR2[1:0]		PUPDR1[1:0]		PUPDR0[1:0]	
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Bits 2y:2y+1 **PUPDRy[1:0]:** Port x configuration bits (y = 0..15)
These bits are written by software to configure the I/O pull-up or pull-down
00: No pull-up, pull-down
01: Pull-up
10: Pull-down
11: Reserved

GPIO port input data register (GPIOx_IDR) (x = A..E and H)

Address offset: 0x10

Reset value: 0x0000 XXXX (where X means undefined)

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Reserved															
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
IDR15	IDR14	IDR13	IDR12	IDR11	IDR10	IDR9	IDR8	IDR7	IDR6	IDR5	IDR4	IDR3	IDR2	IDR1	IDR0
r	r	r	r	r	r	r	r	r	r	r	r	r	r	r	r

Bits 31:16 Reserved, must be kept at reset value.

Bits 15:0 **IDRy**: Port input data (y = 0..15)

These bits are read-only and can be accessed in word mode only. They contain the input value of the corresponding I/O port.

GPIO port output data register (GPIOx_ODR) (x = A..E and H)

Address offset: 0x14

Reset value: 0x0000 0000

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Reserved															
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
ODR15	ODR14	ODR13	ODR12	ODR11	ODR10	ODR9	ODR8	ODR7	ODR6	ODR5	ODR4	ODR3	ODR2	ODR1	ODR0
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Bits 31:16 Reserved, must be kept at reset value.

Bits 15:0 **ODRy**: Port output data (y = 0..15)

These bits can be read and written by software.

Note: For atomic bit set/reset, the ODR bits can be individually set and reset by writing to the GPIOx_BSRR register (x = A..E and H).

GPIO port bit set/reset register (GPIOx_BSRR) (x = A..E and H)

Address offset: 0x18

Reset value: 0x0000 0000

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
BR15	BR14	BR13	BR12	BR11	BR10	BR9	BR8	BR7	BR6	BR5	BR4	BR3	BR2	BR1	BR0
w	w	w	w	w	w	w	w	w	w	w	w	w	w	w	w
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
BS15	BS14	BS13	BS12	BS11	BS10	BS9	BS8	BS7	BS6	BS5	BS4	BS3	BS2	BS1	BS0
w	w	w	w	w	w	w	w	w	w	w	w	w	w	w	w

Bits 31:16 **BRy:** Port x reset bit y (y = 0..15)

These bits are write-only and can be accessed in word, half-word or byte mode. A read to these bits returns the value 0x0000.

0: No action on the corresponding ODRx bit

1: Resets the corresponding ODRx bit

Note: If both BSx and BRx are set, BSx has priority.

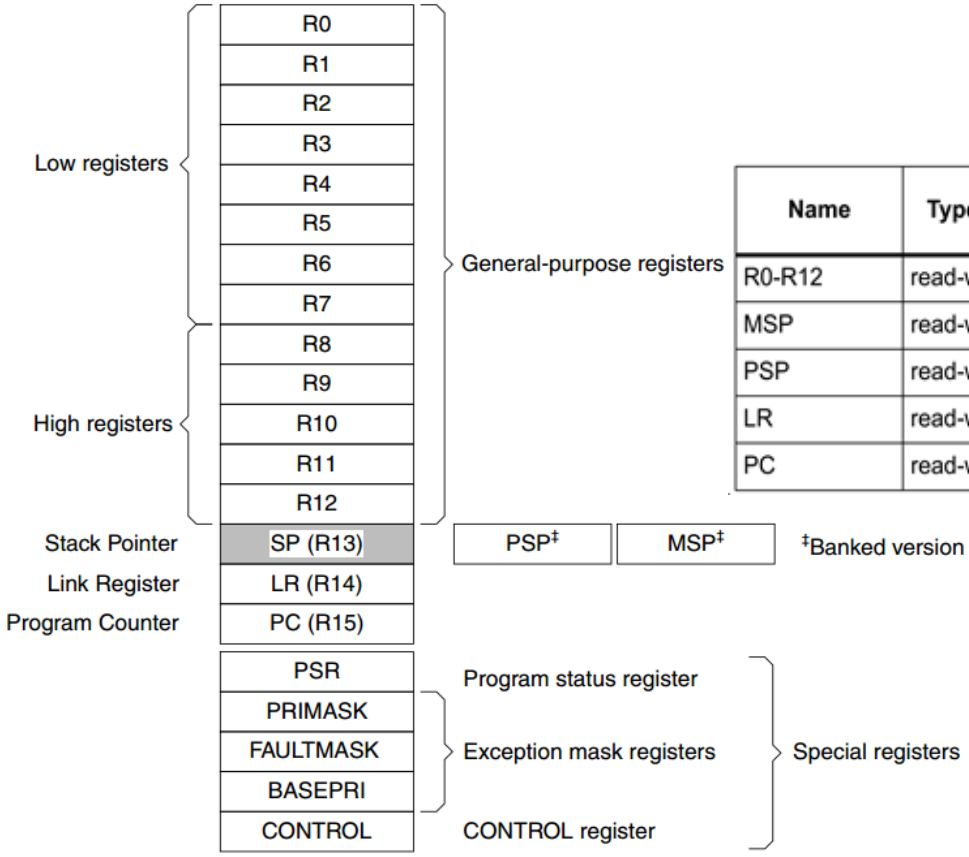
Bits 15:0 **BSy:** Port x set bit y (y= 0..15)

These bits are write-only and can be accessed in word, half-word or byte mode. A read to these bits returns the value 0x0000.

0: No action on the corresponding ODRx bit

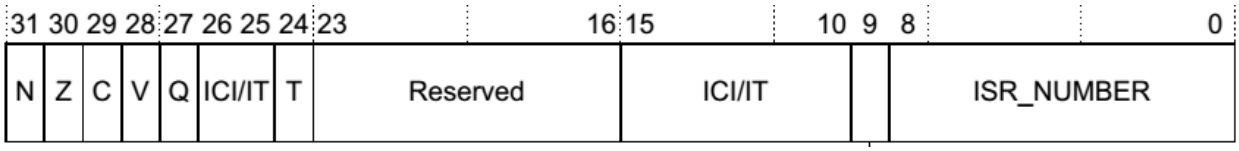
1: Sets the corresponding ODRx bit

2.3. Register



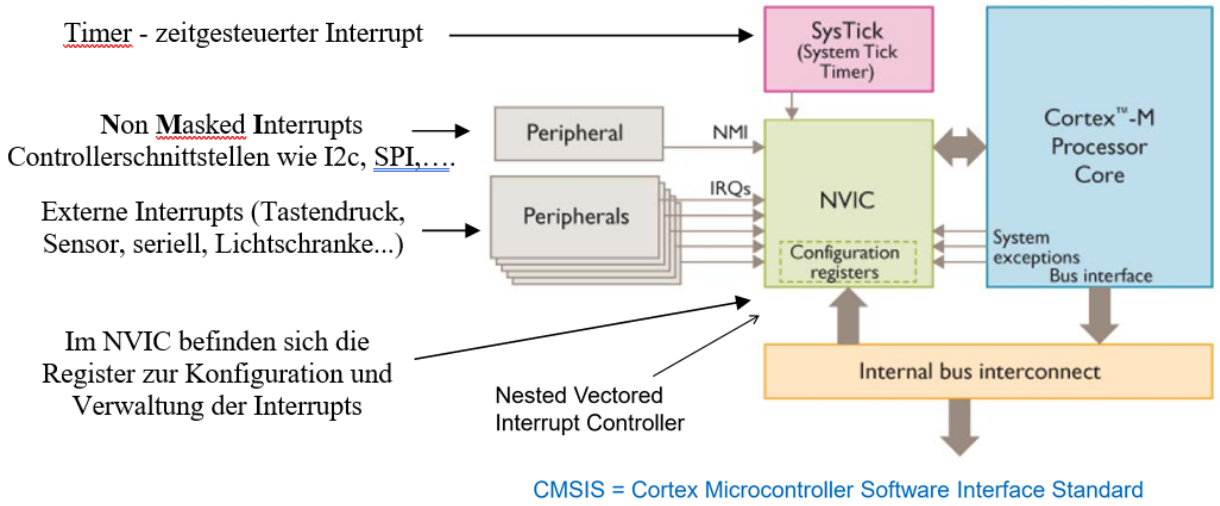
Name	Type(1)	Required privilege(2)	Reset value
R0-R12	read-write	Either	Unknown
MSP	read-write	Privileged	See description
PSP	read-write	Either	Unknown
LR	read-write	Either	0xFFFFFFFF
PC	read-write	Either	See description

R-Nr	Zweck
R0-12	General Purpose Register für Datenoperationen (Load-Store-Arithmethic-Logic)
R13	Stack-Pointer SP
R14	Link-Register LR
R15	Programm Counter PC (halfword aligned-2er Schritte)



PSR	Programm Status Register
Bit 31	N: Negative or less flag 0: Ergebnis Operation positiv, 0, oder größer-gleich als … 1: Ergebnis Operation negativ oder kleiner als …
Bit 30	Z: Zero flag 0: Ergebnis Operation war ungleich Null 1: Ergebnis Operation war gleich Null
Bit 29	C: Carry or borrow flag 0: Addition ergab keinen Übertrag, Subtraktion ergab Ergebnis <0 1: Addition ergab einen Übertrag, Subtraktion ergab Ergebnis >=0
Bit 28	V: Overflow flag 0: Operation ergab keinen Überlauf Wertebereich 1: Operation ergab einen Überlauf Wertebereich

2.4. Interrupts – NVIC (Nested Vectored Interrupt Controller)



CMSIS = Cortex Microcontroller Software Interface Standard

Konfiguration im NVIC

→ Interruptpriorität **NVIC_IPRx → I**nterrupt **P**riority **R**egister für x= 0-80
Setzen der Priorität eines Interrupts zwischen 0 und 255 (8 Bit)
Je kleiner der Wert, desto höher ist die Priorität
Bei gleichem Wert hat der Interrupt mit der kleinsten Exceptionnummer vorrang

→ Interruptfreigabe **NVIC_ISER0 (1)** → **I**nterrupt **S**et **E**nable **R**egister
NVIC_ICER0 (1) → **I**nterrupt **C**lear **E**nable **R**egister
Hier wird festgelegt, ob eine ISR ausgeführt wird oder nicht
bzw. direkt über den NVIC → **IMR** = **I**nterrupt **M**ask **R**egister

→ Interruptanforderung **NVIC_ISPRx** → **I**nterrupt **S**et **P**ending **R**egister für x = 0-2
NVIC_ICPRx → **I**nterrupt **C**lear **P**ending **R**egister

Interrupt Set Enable Register 0 (ISER0)

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Enable Bit	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Interrupt Number	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	I2C1_EV	TIM4	TIM3	TIM2	TIM11	TIM10	TIM9	LCD	EXTI9_5	COMP	DAC	USB_LP	USB_HP	ADC1	DMA1_CH7	DMA1_CH6	DMA1_CH5	DMA1_CH4	DMA1_CH3	DMA1_CH2	DMA1_CH1	EXTI4	EXTI3	EXTI2	EXTI1	EXTI0	RCC	FLASH	RTC_WKUP	TAMPER_STAMP	PVD	WWDG

Interrupt Set Enable Register 1 (ISER1) *Address of ISER1 = Address of ISER0 + 4*

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Enable Bit	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0
Interrupt Number	44	43	42	41	40	39	38	37	36	35	34	33	32																			
	TIM7	TIM6	USB_FS_WKU	RTC_Alarm	EXTI15_10	USART3	USART2	USART1	SPI2	SPI1	I2C2_ER	I2C2_EV	I2C1_ER																			

Interrupt Clear Enable Register 0 (ICER0)

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Clear Enable Bit	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Interrupt Number	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	I2C1_EV	TIM4	TIM3	TIM2	TIM11	TIM10	TIM9	LCD	EXTI9_5	COMP	DAC	USB_LP	USB_HP	ADC1	DMA1_CH7	DMA1_CH6	DMA1_CH5	DMA1_CH4	DMA1_CH3	DMA1_CH2	DMA1_CH1	EXTI4	EXTI3	EXTI2	EXTI1	EXTI0	RCC	FLASH	RTC_WKUP	TAMPER_STAMP	PVD	WWDG

Interrupt Clear Enable Register 1 (ICER1) *Address of ICER1 = Address of ISER0 + 4*

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Clear Enable Bit	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0
Interrupt Number	44	43	42	41	40	39	38	37	36	35	34	33	32																			
	TIM7	TIM6	USB_FS_WKUP	RTC_Alarm	EXTI15_10	USART3	USART2	USART1	SPI2	SPI1	I2C2_ER	I2C2_EV	I2C1_ER																			

2.4.1. Externe Interrupts

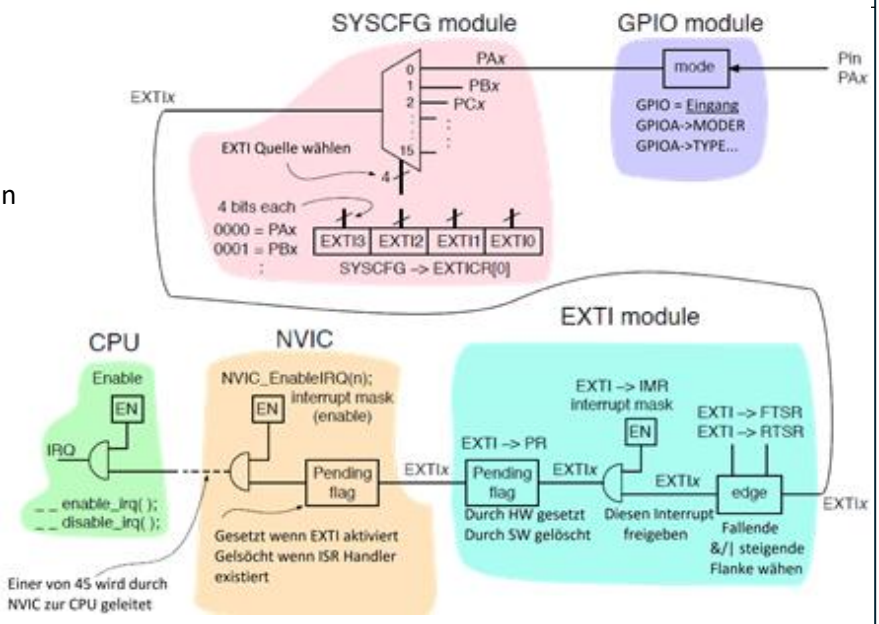
Konfigurationsweg:

→ GPIO-Mode Einstellen
→ _MODER ; _PUPDR ; …
→ Portpin einem ext. Interrupt zuweisen
→ SYSCFG_EXTICRx

→ Trigger-Impuls festlegen (steigende/fallende Flanke)
→ EXTI_FTSR ; EXTI_RTSR

→ Interrup freigeben (maskieren)
→ EXTI_IMR

Interrupt an NVIC, wenn Anforderungs-Flag gesetzt wird (Pendingflag)

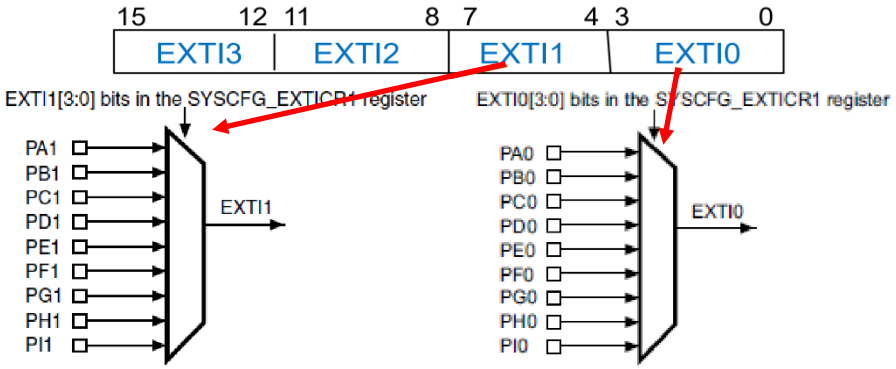


Quelle des externen Interrupts durch die 4 **SYSCFG_EXTICR** Register – **SYSCFG module**

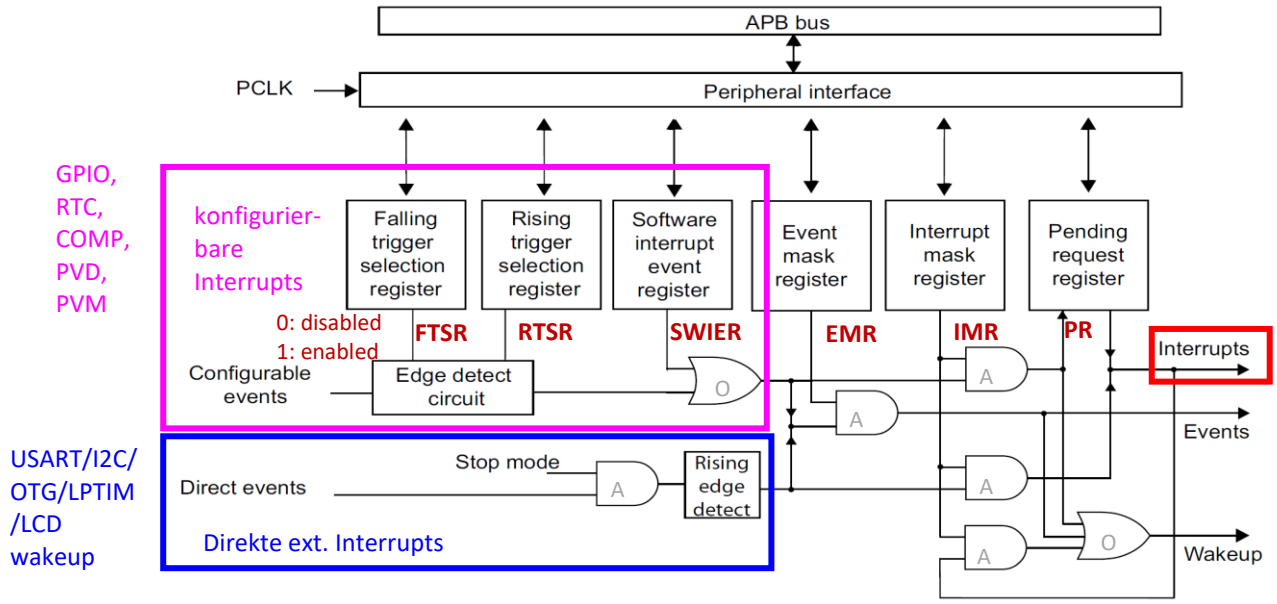
Beispiel: SYSCONFIG_EXTICR**1**

Gleiches gilt für:

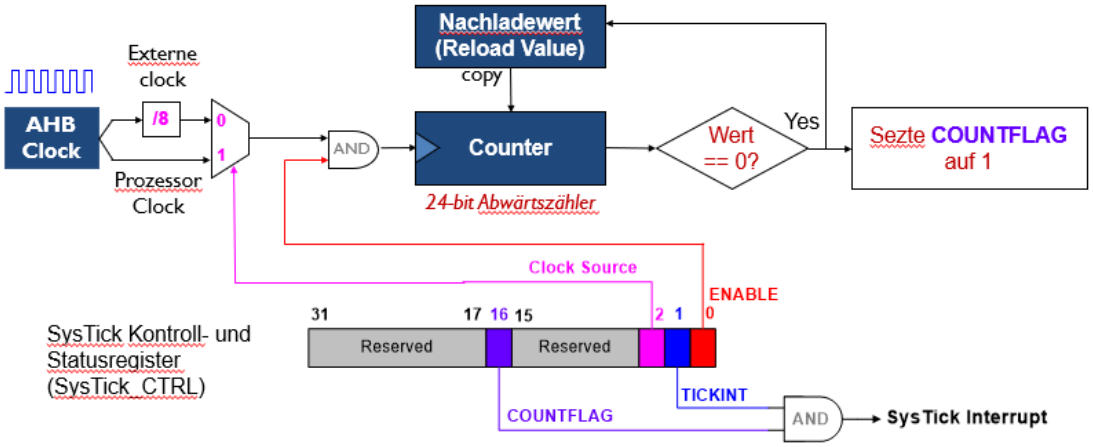
EXTI4-7 (SYSCONFIG_EXTIR2)
EXTI8-11 (SYSCONFIG_EXTIR3)
EXTI12-15 (SYSCONFIG_EXTIR4)



Vom Externen Ereignis zum Interrupt



2.4.2. Timer



SysTick Nachladewertregister (SysTick_LOAD)

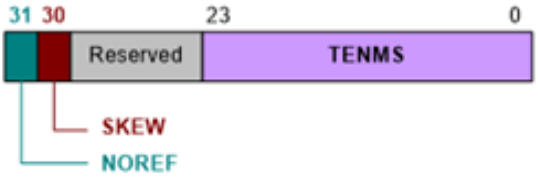
- 24 bits, max.Wert `0x00FF.FFFF` (16,777,215)
- Timer zählt vom Nachladewert bis auf 0 herab.
- Wird RELOAD nach 0 geschrieben, wird SysTick aktiviert, independently of TICKINT
- Zeitintervall zwischen zwei SysTick-Interrupts

  `Intervall = (RELOAD + 1) × Source_Clock_Period`
- Wenn 100 Clock-Perioden zw. 2 SysTicks liegen `RELOAD = 99`

SysTick Aktueller-Wert-Registe (SysTick_VAL)

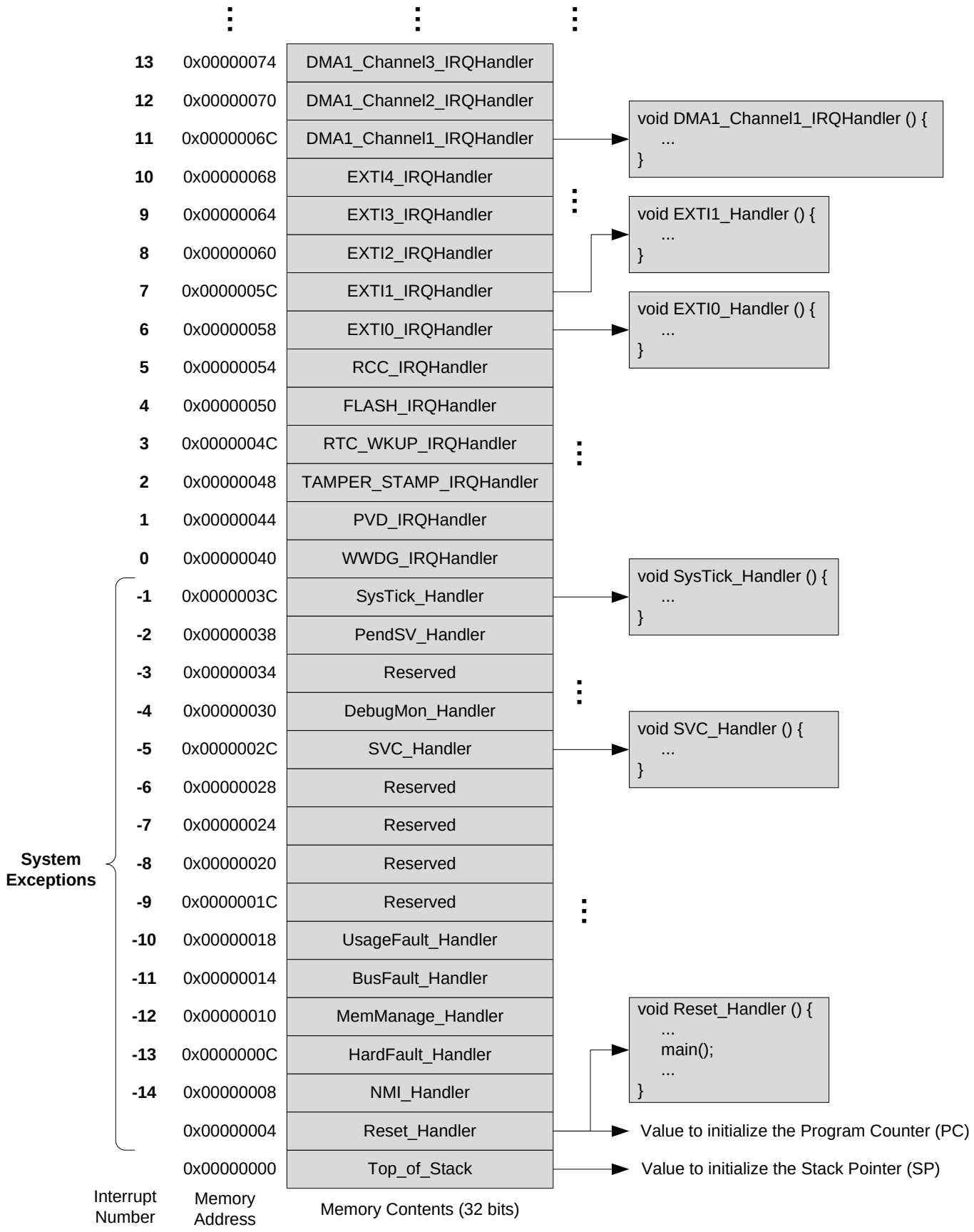
- Gibt den aktuellen Wert des Timer-Zählers aus (wenn man diesen ausliest)
- Wenn es von 1 auf 0 wechselt (letzter Zählwert), wird ein Interrupt ausgelöst
- Zufallswert nach Reset!
  - Muss immer auf 0 gesetzt werdenm bevor ein Timer aktiviert wird!

SysTick Kalibrierungsregister (SysTick_CALIB)



- Ein Read-Only-Register
- TENMS (`10 ms`) hat den Nachladewert, welcher dann alle 10ms übergeben wird.

2.4.3. ISR-Vektor-Tabelle



3. Assembler-Befehle für Cortex-M3/M4 Controller (Auswahl)

Register Lade- und Speicherbefehle			
Befehl	Assembler	Operation	Bemerkung
Register laden	**ldr** rd,=*imm*	rd := *imm*	erzeuge Speicherzelle mit Wert *imm* und lade Wert in Rd
	movs rd,*#imm*	rd := *imm*, Flags NZ := eval(*imm*)	lade *imm* in Rd, 0 ≤ *imm* ≤ 255
	mov rd,*#imm*	rd := *imm*	lade *imm* in Rd, 0 ≤ *imm* ≤ 65535 (32-bit-Opcode)
	movs rd,rm	rd := rm, Flags NZ := eval(rm)	lade Rd mit dem Wert von Rm (Rm wird nicht verändert)
	mov rd,rm	rd := rm	lade Rd mit dem Wert von Rm (Rm wird nicht verändert)
	ldr rd,[rm,*#imm*]	rd := mem[rm+*imm*]	lade Wort, 0 ≤ imm ≤ 124, *imm* Vielfaches von 4
	ldrb rd,[rm,*#imm*]	rd[31:8]:= 0, rd[7:0] := mem[...][7:0]	lade Byte (=8 Bit), 0 ≤ *imm* ≤ 31, vorzeichenlos
	ldrsb rd,[rm,*#imm*]	rd[31:8]:=Vrz, rd[7:0]:=mem[...][7:0]	lade Byte mit Vorzeichenerweiterung, 0 ≤ imm ≤ 31
	ldrh rd,[rm,*#imm*]	rd[31:16]:= 0, rd[15:0] := mem[...][15:0]	lade Halfword (=16 Bit), 0 ≤ imm ≤ 31, vorzeichenlos
	ldrsh rd,[rm,*#imm*]	rd[31:16]:=Vrz, rd[15:0]:=mem[...][15:0]	lade Halfword mit Vorzeichenerweiterung, 0 ≤ imm ≤ 31
Register speichern	**str** rs,[rm,*#imm*]	mem[rm+*imm*]:=rs	speichere Wort, 0 ≤ imm ≤ 124, *imm* Vielfaches von 4
	strb rs,[rm,*#imm*]	mem[...][31:8] gleich, mem[...][7:0]:=rs[7:0]	speichere Byte, 0 ≤ imm ≤ 31, *imm* Vielfaches von 1
	strh rs,[rm,*#imm*]	mem[...][31:16] gleich, mem[...][15:0]:=rs[15:0]	speichere Halfword, 0 ≤ *imm* ≤ 62, *imm* Vielfaches von 2
Stackzugriff	**push** {*reglist*}	Speichern der Inhalte der Register in der Liste auf den Stack	
	pop {*reglist*}	Laden der Register in der Liste mit den Werten auf dem Stack, die dort entfernt werden	
	sub sp,*#imm*	allokiere *imm* Bytes auf dem Stack als Speicher für lokale Variablen, 0 ≤ *imm* ≤ 508, *imm* Vielfaches von 4	
	add sp,*#imm*	gebe *imm* Bytes Speicher auf dem Stack frei, 0 ≤ *imm* ≤ 508, *imm* Vielfaches von 4	
	ldr rd,[sp,*#imm*]	rd := mem[sp+*imm*]	Ladebefehl, Byte und Halfword mit/ohne Vorzeichen möglich
	str rs,[sp,*#imm*]	mem[sp+*imm*]:=rs	Speicherbefehl, Byte und Halfword sind möglich

Arithmetische und logische Befehle			
Addition / Subtraktion	**adds** rd,*#imm*	rd := rd+*imm*, Flags NZCV := eval(rd+*imm*)	addiere zu Rd einen Wert, setze Flags, 0 ≤ *imm* ≤ 255
	adds rd,rm	rd := rd+rm, Flags NZCV := eval(rd+rm)	addiere zu Rd den Wert von Rm, setze Flags
	add rd,*#imm*	rd := rd+rm	addiere Rd mit *imm*, 0 ≤ *imm* ≤ 4095 (32-bit Opcode)
	subs rd,*#imm*	rd := rd-*imm*, Flags NZCV := eval(rd-*imm*)	subtrahiere einen Wert von Rd, setze Flags, 0 ≤ *imm* ≤ 255
	subs rd,rm	rd := rd-rm, Flags NZCV := eval(rd-rm)	subtrahiere den Wert in Rm von Rd, setze Flags
	sub rd,*#imm*	rd := rd-rm	subtr. einen Wert von Rd, 0 ≤ *imm* ≤ 4095 (32-bit Opcode)
Multiplikation	**muls** rd,rm	rd := rd*rm, Flags NZ := eval(rd*rm)	addiere zu Rd einen Wert, setze Flags, 0 ≤ *imm* ≤ 255
Division	**udiv** rd, rm	rd := rd / rm	dividiere Rd durch RM (unsigned)
	sdiv rd, rm	rd := rd / rm	dividiere Rd durch RM (signed, d.h. beachte Vorzeichen)
Schieben (Shift)	**lsl** rd,*#imm*	rd := rd << *imm*	schiebe das Rd um *imm* Bits logisch nach links
	lsr rd,*#imm*	rd := rd >>> *imm* (unsigned)	schiebe das Rd um *imm* Bits logisch nach rechts
	asr rd,*#imm*	rd := rd >> *imm* (signed)	schiebe das Rd um *imm* Bits arithmetisch nach rechts
Bitweise logische Operationen	**ands** rd,rm	rd := rd AND rm, Flags NZ := eval(rd AND rm)	UND-Verknüpfung (s=setze Flags)
	orrs rd,rm	rd := rd OR rm, Flags NZ := eval(rd OR rm)	ODER-Verknüpfung (s=setze Flags)
	eors rd,rm	rd := rd XOR rm, Flags NZ := eval(rd XOR rm)	exklusive ODER-Verknüpfung (s=setze Flags)
	mvns rd,rm	rd := NOT rm, Flags NZ := eval(NOT rm)	Invertierung / 1er Komplement ("move not"), s=set flags
No Operation	**nop**	-	keine Funktion

Sprungbefehle			
Sprünge	**b** *label*	pc := adress(*label*)	springe zu der mit dem *Label* gekennzeichneten Adresse
	bl *label*	pc := adress(label), lr := Rücksprungadresse	Unterprogrammaufruf mit Rücksprungadressspeicherung
	bx lr	pc := lr	Rücksprung (=return) aus einem Unterprogramm
Vergleich	**cmp** rm,*#imm*	Flags NZCV := eval(rm-*imm)*	setze Flags nach Ergebnis der Vergleichssubtraktion
	cmp rm,rn	Flags NZCV := eval(rm-rn)	setze Flags nach Ergebnis der Vergleichssubtraktion
bedingte Sprünge	**beq** *label*	if (Z == 1) branch bzw. (rm == *imm*)	equal: springe, wenn ein "compare" gleich ergibt
	bne *label*	if (Z == 0) branch bzw. (rm != *imm*)	not equal: springe, wenn ein "compare" ungleich ergibt
	bcs *label* / **bhs** *lbl*	if (C == 1) branch bzw. (rm ≥ *imm*)	carry set / "unsigned higher or same" bei vorzeichenlosem Vgl.
	bcc *label* / **blo** *lbl*	if (C == 0) branch bzw. (rm < *imm*)	carry clear / "unsigned lower" bei vorzeichenlosem Vergl.
	bmi *label*	if (N == 1) branch bzw. (rm < 0)	minus = negativ
	bpl *label*	if (N == 0) branch bzw. (rm ≥ 0)	plus = positive or zero
	bvs *label*	if (V == 1) branch bzw. (-2^{31} > rm-*imm* ≥ 2^{31})	signed overflow: Ergebnis verlässt 32-bit-Wertebereich
	bvc *label*	if (V == 0) branch bzw.(-2^{31} ≤ rm-*imm* < 2^{31})	no signed overflow: Ergebnis im 32-bit-Wertebereich
	bhi *label*	if ((C == 1) && (Z == 0)) ... bzw. (rm > *imm*)	"unsigned higher" bei vorzeichenlosem Vergleich
	bls *label*	if ((C == 0) || (Z == 1)) ... bzw. (rm ≤ *imm*)	"unsigned lower or same" bei vorzeichenl. Vergleich
	bge *label*	if (N == V) branch bzw. (rm ≥ *imm*)	"sigend greater or equal"
	blt *label*	if (N != V) branch bzw. (rm < *imm*)	"signed less than"
	bgt *label*	if ((Z == 0) && (N == V)) bzw. (rm > *imm*)	"signed greater than"
	ble *label*	if ((Z == 1) && (N != V)) bzw. (rm ≤ *imm*)	"signed less than or equal"

4. Grundlegende Programmierung mit Mbed

4.1. Pin- und Portdeklaration für Ein-/Ausgaben:

Beispiele:

```
DigitalOut led(LED1);              // LED1 in PinNames.h  vordeklariert   - entspricht PA_5
DigitalIn button(USER_BUTTON);     // USER_BUTTON ist vordeklariert   - entspricht PC_13
InterruptIn taster0(PB_0);         // Portpin PB_0 als Interrupteingang taster0

PortInOut port1(PortC,0x00FF);     // GPIOC Low als Ein- und Ausgabe
PortOut   port2(PortC,0xFF00);     // GPIOC High als Ausgabe, die Bits sind beim Zugriff weiterhin an Position 15:8
PortIn    port3(PortB);            // GPIOB (alle Bits) als Eingabe

// Busse können aus Pins verschiedener Ports bestehen. Sie dürfen nicht in ISRs verwendet werden.
// Tut man es trotzdem, gibt es keine Kompiler- aber Laufzeitfehler, die nur im Monitor erkannt werden!
BusInOut  p4(PA_10, PA_9, PB_9,  PB_8,  PA_15, PA_6,  PA_5,  PA_7);     // GPIO Mixed als Ein- u. Ausgabe
BusIn     p5(PB_7,  PB_6, PB_9,  PB_8,  PA_15, PA_6,  PA_5,  PA_7);     // GPIO Mixed als Eingabe
BusOut    p6(PC_6,  PC_7, PB_9,  PB_8,  PB_12, PB_14, PB_13, PB_15);    // GPIO Mixed als Ausgabe
```

DigitalOut output(PC_0); *// Definiert einen Ausgangspin an PC_0 mit dem Namen output*

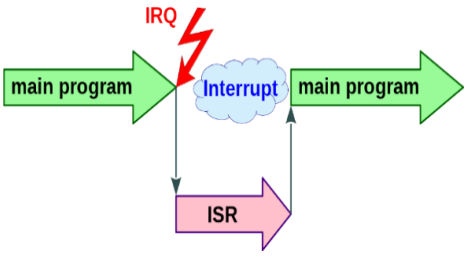
Funktion	Beschreibung	Beispielanwendung
write(…) Kurzform "="	Ausgangspin beschreiben	output.write(1); output = 1;
read() Kurzform "="	Ausgangspin einlesen	int wert = output.read(); wert = output;

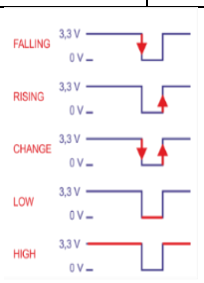
DigitalIn input(PB_0); *// Definiert einen Eingangspin an PB_0 mit dem Namen input*

Funktion	Beschreibung	Beispielanwendung
read() Kurzform "="	Eingangspin einlesen	int eingabe = input.read(); eingabe = input;
mode(…)	Eingangsmodus festlegen (moder-Register) *PullNone* (00), *PullUp* (01), *PullDown* (10), *OpenDrain* (11)	input.mode(PullUp)

InterruptIn taster(PB_0); *// Definiert einen Interrupteingang an PB_0 mit dem Namen taster*

Funktion	Beschreibung	Beispielanwendung
rise(…)	Verbindet eine ISR für Reaktion auf positive Flanke	taster.rise(&isr_taster_rse);
fall(…)	Verbindet eine ISR für Reaktion auf negative Flanke	taster.fall(&isr_taster_fall);
read()	Eingangspin einlesen	int wert = taster.read();
mode(…)	*PullNone* (00), *PullUp* (01), *PullDown* (10), *OpenDrain* (11)	taster.mode(PullDown);
enable_irq()	Interrupt freigeben	taster.enable_irq();
disable_irq()	Interrupt sperren	taster.disable_irq();





PortOut outport(PortC,0x00FF); *// Definiert 8-Bit Ausgangsport an Port C mit dem Namen outport*

Funktion	Beschreibung	Beispielanwendung
write(…) Kurzform "="	Ausgangsport beschreiben	outport.write(0x34); outport = 0x34;
read() Kurzform "="	Werte an den Pins des Ausgangsports einlesen	int wert = outport.read(); wert = outport;

PortIn inport(PortB,0x00FF); *// Definiert 8-Bit Eingangsport an Port B mit dem Namen inport*

Funktion	Beschreibung	Beispielanwendung
mode(…)	Setze Pinmodus *PullUp*, *PullDown*, *PullNone* oder *OpenDrain*	inport.mode(*PullUp*);
read() Kurzform "="	Werte an den Pins des Eingangsports einlesen	int wert = inport.read(); wert = inport;

PortInOut ioport(PortC,0xFF00); *// Definiert 8-Bit bidirektionalen Port am High-Part von Port C mit dem Namen ioport*

Funktion	Beschreibung	Beispielanwendung
mode(…)	setze Pinmodus *PullUp*, *PullDown*, *PullNone* oder *OpenDrain*	ioport.mode(*PullUp*);
input()	konfiguriere Port-Pins als Eingang	ioport.input();
read() Kurzform "="	Werte an den Pins des Eingangsports einlesen	int wert = ioport.read() >> 8; wert = ioport >> 8;
output()	konfiguriere Port-Pins als Ausgang	ioport.output();
write(…) Kurzform "="	Port beschreiben	ioport.write(0x34 << 8); ioport = 0x34 << 8;

4.2. Wartefunktionen

Funktion	**Beschreibung**	Beispielanwendung
wait_us(…)	warte x Mikrosekunden mit x als int	wait_us(7000);
wait_ns(…)	warte x Nanosekunden mit x als unsigned int	wait_ns(800);
thread_sleep_for(…)	schlafe x Millisekunden mit x als unsigned int	thread_sleep_for(20);

4.3. Timerimplementierung

Timer stoppuhr; // definiert einen Timer mit dem Namen stoppuhr

Funktion	Bedeutung	Beispielanwendung
start()	Startet den Timer	stoppuhr.start( );
stop()	Stoppt den Timer	stoppuhr.stop( );
reset()	Resetet den Timer auf 0	stoppuhr.reset( );
elapsed_time()	liefert ein *std::chrono::microseconds*-Objekt zurück, von dem man mit *count()* die µs auslesen kann.	int us = stoppuhr.elapsed_time().count();
Abgekündigte Funktionen (engl. deprecated methods), die aber noch funktionieren:		
read()	Gibt die Zeit als float in Sekunden zurück	if((float sek= stoppuhr.read())>=20)
read_ms()	Gibt die Zeit als integer in Millisekunden zurück	int ms = stoppuhr.read_ms( );
read_us()	Gibt die Zeit als integer in Mikrosekunden zurück	int us = stoppuhr.read_us( );

Timeout impuls; // definiert einen Timeout mit dem Namen impuls

Funktion	Bedeutung	Beispielanwendung
attach(…)	Verbindet eine ISR, die nach Ablauf von x Zeiteinheiten aufgerufen wird (x = *std::chrono*-Objekt, z.B. 2s, 4ms, 6us).	impuls.attach(&isr_timeout,20us);
detach()	Löst die Funktion vom Timout	impuls.detach( );
Abgekündigte Funktionen (engl. deprecated methods), die aber noch funktionieren:		
attach(…)	Verbindet eine ISR, die bei Timout nach x Sekunden aufgerufen wird. (x = *float*)	impuls.attach(&isr_pulse_end,2.5);
attach_us(…)	Verbindet eine ISR, die bei Timout nach x Mikrosekunden aufgerufen wird. (x = *int*).	impuls.attach_us(&isr_pulse_end,7);

Ticker wiederholer; // definiert einen Ticker mit dem Namen wiederholer

Funktion	Bedeutung	Beispielanwendung
attach(…)	Verbindet eine ISR, die vom Ticker alle x Zeiteinheiten aufgerufen wird (x = *std::chrono*-Objekt, z.B. 2s, 4ms, 6us).	wiederholer.attach(&isr_blink,500ms);
detach()	Löst die Funktion vom Ticker	wiederholer.detach( );
Abgekündigte Funktionen (engl. deprecated methods), die aber noch funktionieren:		
attach(…)	Verbindet eine ISR, die alle x Sekunden aufgerufen wird (x = *float*).	impuls.attach(&isr_blink,2.5);
attach_us(…)	Verbindet eine ISR, die alle x µs aufgerufen wird (x = *int*).	impuls.attach_us(&isr_wave,7);

5. Peripherieunterstützung durch Mbed

5.1. AD-Wandler – Analoge Eingänge

AnalogIn temperatur(PA_0); // PA_0 als analoger Eingang

Funktion	**Bedeutung**	Beispielanwendung
read( )	Liest die Eingangsspannung als float zwischen (0.0 – 1.0) bezogen auf Vcc	float wert = temperatur.read( );
read_u16( )	Liest die Eingangsspannung als unsigned short zwischen (0x0-0xFFF) - 12 Bit	int wert = temperatur.read_u16( );

Mögliche Pins: PA_0 bis PA_7 ; PB_0 bis PB_1 ; PC_0 bis PC_5, PB_0 bis PB_1

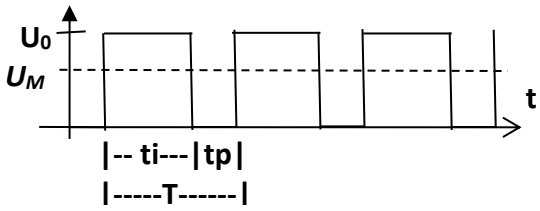
Die Analogeingänge des **µC STM32F411** haben eine **12-Bit-Auflösung** bei **3,3 V** Bezugsspannung.

Beispiel: int wert = temperatur.read_u16( ); // gemessene Spannung = 3,3V * (wert / 4095)
float wert = temperatur.read( ); // gemessene Spannung = 3,3V * wert

5.2. Puls-Weiten-Modulation - PWM-Ausgang

Das Verhältnis der **Impulszeit ti** zur **Periodenzeit T** (den Duty-Cycle) wird als **Tastgrad g** bezeichnet. Entsprechend der Formel ist g ein Wert zwischen 0 und 1.

$$g = t_i/T = t_i / (t_i + t_p) \rightarrow U_M = g * U_0$$



PwmOut pwm(PC_8); // Definiert den Pin PC_8 als PWM-Ausgang

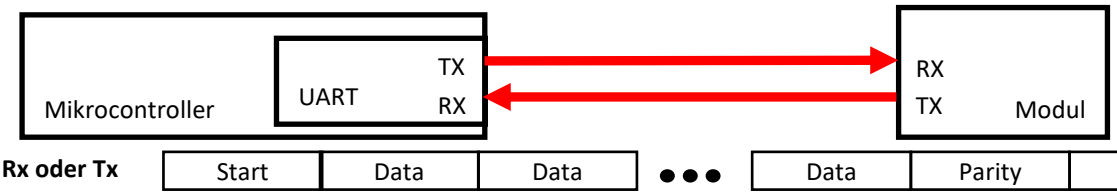
Funktion	**Bedeutung**	**Beispielanwendung**
write(…) *Operator=*	Gibt den Tastgrad g des PWM-Signals an (0.0 – 1.0) oder mit der Kurzform	pwm.write(0.3) pwm = 0.3;
read() *Operator float()*	Liest den Tastgrad g des PWM-Signals (0.0 – 1.0) oder mit der Kurzform	g = pwm.read( ); g = pwm;

period(…)
period_ms(…)
period_us(…)
	Setzt die Periodenzeit T in Sekunden (float),
	Millisekunden (int),
	Mikrosekunden (int).
		pwm.period(0.38);
		pwm.period_ms(43);
		pwm.period_us(70);

pulsewidth(…) pulsewidth_ms(…) pulsewidth_us(…)	Setzt die Impulsbreite ti in Sekunden (float), Millisekunden (int), Mikrosekunden (int).	pwm.pulsewidth(1.58); pwm.pulsewidth_ms (43); pwm.pulsewidth_us (70);

Mögliche Pins: PA_0 bis PA_3 ; PA_5 bis PA_11 ; PA_15
PB_0 bis PB_1 ; PB_3 bis PB_10 ; PB_13 bis PB_15
PC_6 bis PC_9

5.3. UART-Schnittstelle - Universal Asynchronous Receiver Transmitter



Frame: Eine UART-Übertragung beginnt immer mit einem Startbit (Low).
Darauf folgen: → 5-8 **Datenbits, LSB first** (Standard = 8 Bit)
→ 0 oder 1 **Paritybit** (Standard = 0 Bit)
→ 1 oder 2 **Stopbits** (High) (Standard = 1 Bit)
Falls ein Parity-Bit programmiert wurde, kann es gerade Parity (even) oder ungerade Parity (odd) sein.

BufferedSerial monitor(CONSOLE_TX, CONSOLE_RX); *// C..TX und RX sind PA_2, PA_3 in Pinnames.h*
UnbufferedSerial uart(PA_9, PA_10); *// „low-level" Umsetzung ohne Zwischenpuffer*

Funktion	**Bedeutung**	Beispielanwendung
set_baud(…)	Übertragungsrate (bit/Sekunde fest)	monitor.set_baud(9600);
set_format(…)	Übertragungsformat - Bitanzahl, Parität, Stopbitanzahl (1 oder 2)	monitor.set_format(8,SerialBase::None,1); monitor.set_format(8,SerialBase::Parity(1),2);
write(…)	Sendet eine Anzahl von Bytes aus einem Datenpuffer.	int nBytes = monitor.write(buffer,12);
read(…)	Liest empfangene Bytes in einen Datenpuffer.	int nBytes =monitor.read(buffer,bufsize);
Methoden, die es nur in *BufferedSerial* gibt:		
readable()	Abfrage, ob es Zeichen zum Einlesen gibt.	bool flag = monitor.readable( );
writable()	Abfrage, ob im Puffer Platz für ein Zeichen ist.	bool flag = monitor.writable( );
Methoden, die es nur in *UnbufferedSerial* gibt:		
attach(…)	Funktionsaufruf, bei Interrupt beim **Empfang** der seriellen Schnittstelle beim **Senden**	uart.attach(&isr_in, Serial::RxIrq); uart.attach(&isr_out, Serial::TxIrq);

Hilfsfunktion zur Textformatierung: int snprintf (char* buffer, size_t buf_size, const char* format, ...):
Beispiel: char buffer[32]; snprintf(buffer,32,"%d: %s\n",42,"Nachricht"); monitor.write(buffer,strlen(buffer));

Steuerzeichen zur Textformatierung bei Verwendung von printf

\b	BS (backspace) – setzt den Cursor um eine Position nach links.
\f	FF(formfeed) – ein Seitenvorschub wird ausgelöst.
\n	NL (newline) – der Cursor geht in die nächsten Zeile.
\r	CR (carriage return) – der Cursor springt zum Anfang der aktuellen Zeile.
\t	HT (horizontal tab) – Zeilenvorschub zur nächsten Tabulatorposition (meistens 8 Leerzeichen)
\v	VT (vertical tab) – der Cursor springt zur nächsten vertikalen Tabulatorposition.
\"	" wird ausgegeben.
\'	' wird ausgegeben.

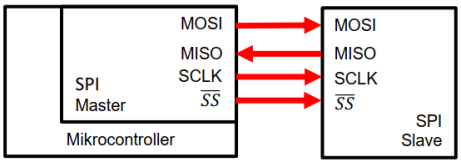
Variablenübergabe mit %-Operator bei Verwendung von printf

%i	Integer	%5i gibt die Zahl mindestens 5-stelig aus (führende Nullen sind Leerzeichen)
%f	float	%6.2f gibt die Zahl mit mind. 6 Stellen inklusive . und 2 Nachkommastellen aus (3+1+2)
%c	character	

Beispiel:
```
int a = 4; float f = 12.3; char  ch ='B';
printf(" Zahl a ist %6i \n\r Zahl f ist %6.2f \n\r Zeichen ch ist %c",a,f,ch);
```

Ausgabe:
```
Zahl a ist      4
Zahl f ist  12.30
Zeichen ch ist B
```

5.4. SPI-Interface - Serial Peripheral Interface

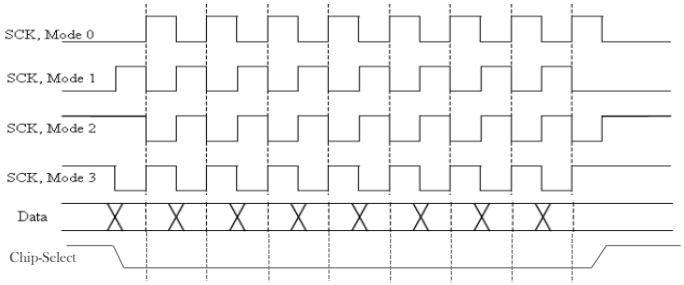


Jeder Teilnehmer benötigt mindestens drei gemeinsamen Verbindungsleitungen:

SCK → Serial Clock (SLK), wird vom Controller zur Synchronisation ausgegeben
MISO → Master Input, Slave Output oder POCI → peripheral out/controller in oder SDO → Serial Data Out
MOSI → Master Output, Slave Input oder PICO → peripheral in/controller out oder SDI → Serial Data In

und eine Auswahlleitung pro Slave, die mit einer logisch 0 den jeweiligen Slave auswählt.
SS → Slave Select oder CS → Chip Select oder CE → Chip Enable oder STE → Slave Transmit Enable

SPI ist **Vollduplexfähig** und hat die folgenden Einstellmöglichkeiten für 4 Übertragungsmodi, mit denen die Flanken für Polarität und Phase eines Signals, das ausgegeben oder eingelesen wird, ausgewählt werden kann.



Mode	Polarity	Phase
0	0	0
1	0	1
2	1	0
3	1	1

SPI spi_master(SPI_MOSI , SPI_MISO , SPI_SCK); // SPI-Master an PA_7 , PA_6 , PA_5

Funktion	**Bedeutung**	Beispielanwendung
format(…)	Legt Datenlänge und Übertragungsmode (0-3) fest.	spi_master.format(8,1); // 8 Bit mode 1
frequency(…)	Taktfrequenz der Übertragung in Hz	spi_master.frequency(1000);
write(…)	Schreibbefehl zu SPI Slave mit Empfangswert	read_dat=spi_master.write(send_dat);

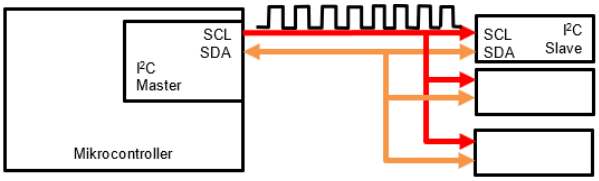
Anmerkung: SPI_CS (Chip-Select, = SPI_SS) wird in der Regel mit einem normalen *DigitalOut*-Pin realisiert.

SPISlave spi_slave(SPI_MOSI , SPI_MISO , SPI_SCK ,NC); // SPI-Slave an PA_7 , PA_6 , PA_5

Funktion	**Bedeutung**	**Beispielanwendung**
format(…)	Legt Datenlänge und Übertragungsmode (0-3) fest.	spi_slave.format(8,1); // 8 Bit mode 1
frequency(…)	Taktfrequenz der Übertragung in Hz	spi_slave.frequency(1000);
receive(…)	Prüft ob Daten in Empfangspuffer vorhanden sind	if (spi_slave.receive( )) { …. }
read(…)	Liest daten aus dem Empfangspuffer als Slave	int read_dat=spi_slave.read( );
reply(…)	Legt den Wert fest, der beim nächsten Schreibebefehl des Masters zurückgeliefert wird.	spi_slave.reply(slave_dat);

5.5. I2C-Interface - Inter-Integrated Circuit-IF

SCL (Serial Clock): Taktleitung
SDA (Serial Data): Datenleitung



I2C i2c(I2C_SDA, I2C_SCL); // I2C-Master an PB_9, PB_8

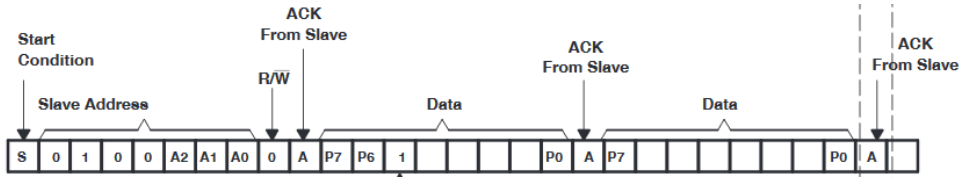
Funktion	**Bedeutung**	Beispielanwendung
frequency(…)	Taktfrequenz der Übertragung in Hz	i2c.frequency(1000);
read(…)	Liest die Anzahl von Datenbytes vom adressierten Slave in den adressierten Datenspeicher.	i2c.read(i2c_adr,dat_adr,6,false); // (Slaveadresse, Datenzeiger,Byteanzahl)
write(…)	Schreibt die Anzahl der adressierten Datenbytes auf den adressierten Slave.	i2c.write(i2c_adr, dat_adr, 8, false); // (Slaveadr., Datenzeiger,Byteanzahl)
start( )	Erzeugt eine start condition auf I2C Bus	i2c.start( );
stop( )	Erzeugt eine stop condition auf I2C Bus	i2c.stop( );

I2CSlave i2c_sl(I2C_SDA, I2C_SCL); // I2C-Slave an PB_9, PB_8

Funktion	**Bedeutung**	Beispielanwendung
frequency(…)	Taktfrequenz der Übertragung in Hz	i2c_sl.frequency(1000);
address(…)	Setzt die I2C-Slave-Adresse	i2c_sl.address(0x7E);
read(…)	Liest die Anzahl von Datenbytes vom Master in den adressierten Datenspeicher.	int n = i2c.read(dat_adr,6); // (Datenzeiger,Byteanzahl)
write(…)	Schreibt die Anzahl der adressierten Datenbytes auf den adressierten Master	int error_nr = i2c_sl.write(dat_adr, 8); // (Datenzeiger,Byteanzahl)
stop( )	Versetzt den I2C Slave in den bekannten Empfangsstatus.	i2c_sl.stop( );

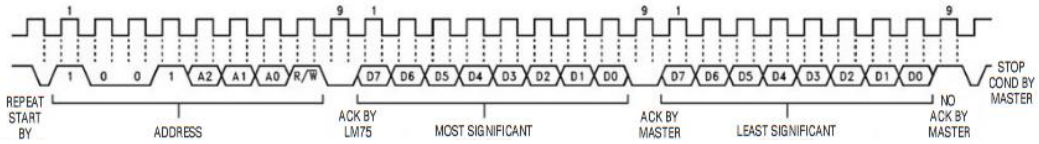
Beispiel:
schreiben zum I2C-IC

PCF 8574



Beispiel:
Lesen vom I2C-IC

LM 75:



Quelle:
www.alldatasheet.com

UPPER BYTE								LOWER BYTE							
D15	D14	D13	D12	D11	D10	D9	D8	D7	D6	D5	D4	D3	D2	D1	D0
Sign bit 1= Negative 0 = Positive	MSB 64°C	32°C	16°C	8°C	4°C	2°C	1°C	LSB 0.5°C	X	X	X	X	X	X	X

X = Don't care.

Weitere Daten zum LM75

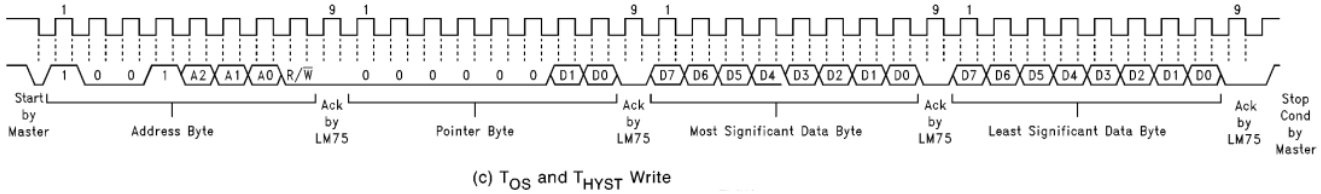
Pointerregister: das erste Byte aller Schreibbefehle setzt dieses Register, das eines der 4 anderen Register für Operationen auswählt:

P7	P6	P5	P4	P3	P2	P1	P0
0	0	0	0	0	0	Register Select	

0x00 Temperaturregister (read)
0x02 THyst (read / write)
0x03 Tos (read / write)

D15	D14	D13	D12	D11	D10	D9	D8	D7	D6	D5	D4	D3	D2	D1	D0
MSB	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	LSB	X	X	X	X	X	X	X

Ein-Austemperaturen Einschreiben – write → R/W=0



(c) T_{OS} and T_{HYST} Write

0x04 Konfigurationsregister (read / write)

D7	D6	D5	D4	D3	D2	D1	D0
0	0	0	Fault Queue		O.S. Polarity	Cmp/Int	Shutdown

Power up default is with all bits "0" (zero).

D0: Shutdown: When set to 1 the LM75 goes to low power shutdown mode.

D1: Comparator/Interrupt mode: 0 is Comparator mode, 1 is Interrupt mode.

D2: O.S. Polarity: 0 is active low, 1 is active high. O.S. is an open-drain output under all conditions.

D3–D4: Fault Queue: Number of faults necessary to detect before setting O.S. output to avoid false tripping due to noise:

6. Hochsprache C/C++

6.1. Datentypen

Datentyp	Bits	Vorzeichen	Wertebereich
unsigned char	8	+	0 .. 255
(signed) char	8	- +	-128 ..127
uint_32t/uint16_t	32/16	+	0 .. 4.294.967.295 bzw. 0 .. 65.535
int_32t/int16_t	32/16	- +	-2.147.483.648 .. 2.147.483.647 bzw. 32.768 .. 32.767
long	32	+	0 .. 4.294.967.295
float	32	- +	-3,4E38 .. 3,4E38
enum Aufzählungstyp			enum {AUTOMATK, HAND} Zustand = AUTOMATIK;

6.2. Zeiger und Referenzen

```
int x=127;          //Wert
int *y;             //Zeiger
*y=x;               //der Zeiger weist auf eine Variable mit dem Wert von x
y=&x;               //der Zeiger bekommt die Adresse der Variable x im Speicher
```

Beispiel:

	Adresse	RAM
x	0x20000000	127
y	0x20000004	0x20000000

printf("%d %x %d\r\n",x,(int)y,*y); // => liefert folgende Ausgabe: 127 0x20000000 127

6.3. Operatoren

Mathematische Operatoren	
++	Inkrement
--	Dekrement
-	Vorzeichen
*	Multiplikation
/	Division
%	Modulo, Rest der Division
+	Plus
-	Minus

Priorität

Höchste

Niedrigste

Vergleichs- und logische Operatoren	
!	NOT
>	Größer
>=	Größer gleich
<	Kleiner
<=	Kleiner gleich
==	Gleich
!=	Ungleich
&&	AND
||	OR

Da ein Gleichheitszeichen in C ein Zuweisungsoperator ist, weist man z.B. mit x = 10; der Variablen x den Wert 10 zu.

Bitweise Operatoren	
&	UND
|	ODER
^	EXOR
~	Einerkomplement
<<	Nach links schieben
>>	Nach rechts schieben

Kurzschreibweisen	
+=	x += 3; wie x = x + 3
-=	x -= 3; wie x = x - 3
*=	x *= 5; wie x = x * 5
/=	x /= 7; wie x = x / 7

6.4. Schleifen

6.4.1. FOR-Schleife (zählergesteuerte Schleife)

Schleife, mit einer genau berechenbaren Anzahl an Wiederholungen.

```
for (<zaehlvariable=startwert>;<bedingung>;<schrittweite>) {
   …
}
```

- startwert Anfangswert der Zählvariablen
- bedingung Schleife wird so lange durchlaufen, wie die Bedingung wahr ist
- schrittweite Anweisung zum Erhöhen oder Erniedrigen der Zählvariablen

Beispiel:

```
// DigitalOut ausgang(PC_0) 10x invertieren
for (int i=0; i<10; i++) {
  ausgang = !ausgang;
}
```

6.4.2. WHILE-Schleife (kopfgesteuerte Schleife)

Schleife, die wiederholt wird, so lange die Bedingung am Schleifenanfang erfüllt ist.

```
while (<bedingung>) {
   …
}
```

Solange die am Anfang stehende **Bedingung erfüllt ist,** wird die Schleife wiederholt. Die Prüfbedingung steht **vor den Anweisungen**, sie heißt deshalb **kopfgesteuerte Schleife.**

Wenn die am Schleifenanfang stehende **Bedingung nicht erfüllt ist,** dann wird die gesamte Schleife übersprungen.

Beispiel:

```
// Solange der Taster DigitalIn taster(PA_1) gedrückt ist, wird der
// Ausgang DigitalOut ausgang(PC_0) invertiert
while (taster==true) {
  ausgang = !ausgang;
}
```

6.4.3. Do-WHILE-Schleife (fußgesteuerte Schleife)

Schleife, die mindestens einmal durchlaufen wird, da erst am Ende der Schleife mit der Überprüfung der Bedingung entschieden wird, ob die Schleife wiederholt werden muss.

```
do {
   …
} while (<bedingung>);
```

Beispiel:

```
// Die Schleife wird maximal 100 mal und minimal 1 mal durchlaufen. Sie wird früh-
// zeitig abgebrochen, wenn der Taster DigitalIn taster(PA_1) gedrückt (=1) wird.
x = 100;
do {
  x--;
} while ((x>0) && taster==0);
```

6.5. Programmverzweigungen

6.5.1. Einfache Verzweigung mit if

Bei der if-Anweisung werden die Anweisungen innerhalb des if-Blocks nur dann ausgeführt, falls die Bedingung wahr ist.

```
if (<bedingung>) {
   <anweisung1>;
   <anweisung2>;
   …
}
```

Beispiel:

```
// Wenn taster1 gedrückt ist, soll ausgang1 eins und ausgang2 null werden.
// Drückt man dagegen taster2, wird nur ausgang2 zu eins.
if (taster1==1) {
  ausgang1 = 1;     // Block mit mehreren Anweisungen wird ausgeführt,
  ausgang2 = 0;     // wenn die Bedingung hinter if wahr ist
}
if (taster2==1)
  ausgang2 = 1;     // Nur eine Anweisung, keine {} notwendig
```

6.5.2. Zweiseitige Verzweigung mit if

Bei der if/else-Anweisung kann zwischen **zwei Alternativen** entschieden werden. Ist die Bedingung wahr, so wird die erste Alternative (if-Block), ansonsten die zweite Alternative (else-Block) an Anweisungen ausgeführt.

```
if (<bedingung>) {
   <anweisung1>;
   <anweisung2>;
   …
} else {
   <anweisung3>;
   <anweisung4>;
   …
}
```

Beispiel:

```
// Wenn taster1 gedrückt ist, soll ausgang1 eins und ausgang2 null werden,
// andernfalls soll ausgang1 null und ausgang2 eins werden.
if (taster1==1) {
  ausgang1 = 1;     // Block mit mehreren Anweisungen wird ausgeführt,
  ausgang2 = 0;     // wenn die Bedingung hinter if wahr ist
} else {
  ausgang1 = 0;     // Block mit mehreren Anweisungen wird ausgeführt,
  ausgang2 = 1;     // wenn die Bedingung hinter if nicht wahr ist
}
```

6.5.3. Mehrere Verzweigungen mit if

```
if (<bedingung1>) {
   <anweisung1>;
   …
} else if (<bedingung2>) {
   <anweisung2>;
   …
} else {
   <anweisung3>;
   …
}
```

6.5.4. Fallunterscheidung mit switch

Mit der switch-Anweisung kann aus einer **Reihe von Alternativen** ausgewählt werden. Es ist zulässig, dass mehrere Möglichkeiten gültig sind und dieselbe Wirkung haben. Sie werden nacheinander aufgelistet. Passt keine der Möglichkeiten, dann wird die **default**-Einstellung ausgeführt. Achtung! Auf keinen Fall **break** vergessen!!!

```
switch (<vergleichswert>) {
   case <wert1>:      <anweisung1>;      <anweisung2>;      …      break;
   case <wert2>:      <anweisung3>;      <anweisung4>;      …      break;
   …
   default:      <anweisung5>;      … break;
}
```

Beispiel:

```
// In der Variablen ergebnis ist ein Messergebnis oder eine Zahl gespeichert.
// Abhängig vom genauen Wert sollen nun bestimmte Reaktionen erfolgen.
switch (ergebnis) {
  case 0x10:
  case 0x20:    ausgang1 = 1;    break;
  case 0x30:    ausgang1 = 0;    break;
  default:    ausgang2 = 1;    break;
}
```

Hinweis: **Switch-Variablen** müssen einen **einfachen Datentyp** verwenden. Hinter **case** müssen **Konstanten** stehen. Diese können mit #define am Anfang des Programms deklariert werden.

Beispiel:

```
# define RECHTS  0x10    // ohne Semikolon!!
# define LINKS  0x20
# define RECHTSKURVE 0b0100
# define LINKSKURVE 0b1000

unsigned char richtung;
…
switch (richtung) {
  case RECHTS:    motor = RECHTSKURVE;    break;
  case LINKS:     motor = LINKSKURVE;     break;
  default:    motor = vorwaerts;    break;
}
```

6.6. Operationen (Unterprogramme, Funktionen)

Beispiele:

Deklaration von Operationen

```
void addieren(void);                    // ohne Rückgabewert, ohne Parameter
void zeitms(int msec);                  // ohne Rückgabewert, mit Parameter
float berechneQuadrat(float pQ);        // mit Rückgabewert, mit Parameter
```

Definition von Operationen

```
int a, result;                          // globale Variablen
void addieren(void) {                   // Operationsname
   result = a + a;                      // Anweisung(en)
}
```

Operationen mit Übergabewert

```
void zeitms(int msec) {                 // Übergabewert msec
   int t1;                              // lokale Variable
   for (t1=msec; t1 > 0; t1--) {
     wait_us(1000);                     // Zeitschleife;
   }
}
```

Operationen mit Rückgabewert

```
float berechneQuadrat(float pQ=10) { // Parameter mit Standardwert
  return pQ*pQ;                      // Rückgabewert
}
```