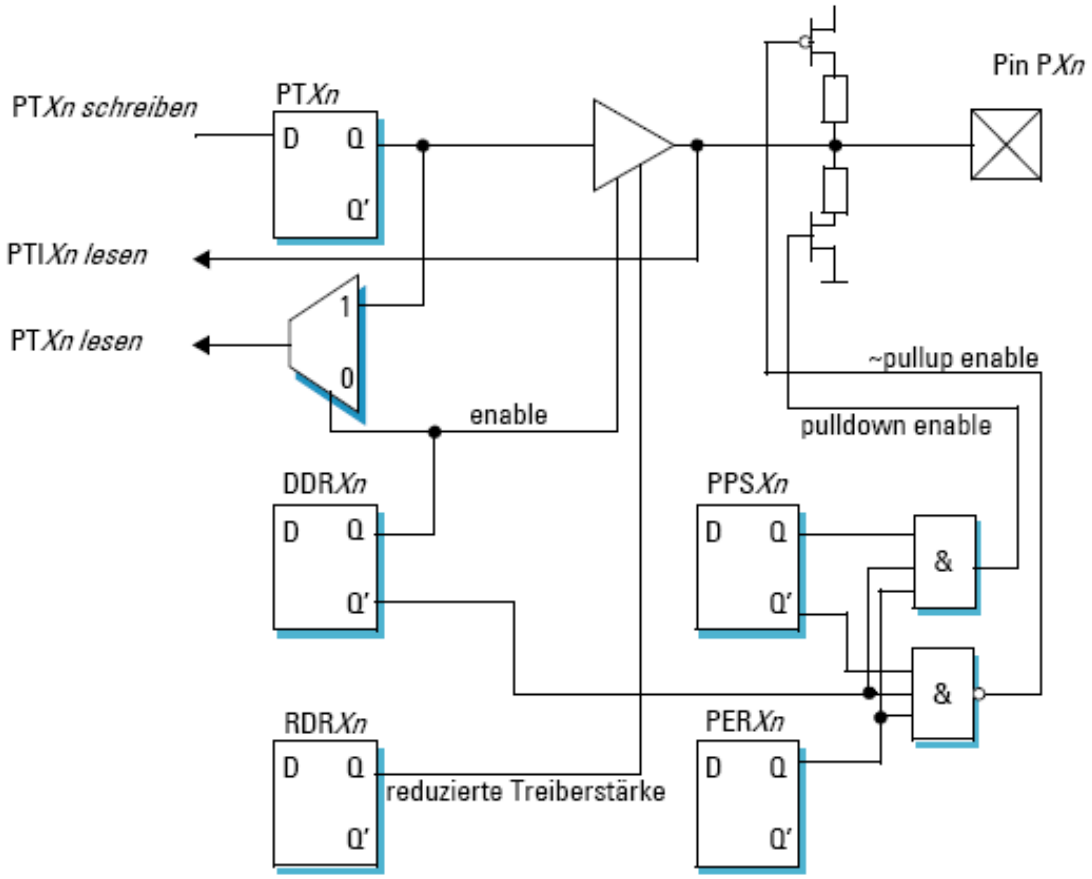
Kapitel 3
Peripheriekomponenten, Ein-Ausgabeprogrammierung, Interrupts

3.1 Digital-Ein- und Ausgänge (General Purpose Input/Output GPIO, siehe [3.3 und 3.10])



Schema eines einzelnen Digital-Ein/Ausgangs Pin x.n
(DDRx.n = 1 → Pin n des Ports X arbeitet als Ausgang, mit n=0…7 und x=A, B, E, H, J, K, M, P, S, T)

- Die CPU verfügt über mehrere Gruppen von jeweils 8 Digital-Ein-/Ausgängen (Ports), die bitweise über das jeweilige Datenrichtungsregister **DDRx** (Data Direction Register) als Ein- oder Ausgang konfiguriert werden können.
- Lesen und Schreiben des Port-Pins erfolgt über das Datenregister **PTx** bzw. **PORTx** mit MOVB-Befehlen für alle 8bits oder mit BSET/BCLR zum Schreiben bzw. BRCLR/BRTST zum Testen einzelner Pins.
- Über das Register **RDRx** (Reduced Driver kann die Treiberstärke eines Ausgangs reduziert werden (um die EMV-Eigenschaften zu verbessern), mit **PPSx** (Port Polarity Select) kann ein Pull-Up- bzw. Pull-Down Widerstand ausgewählt und mit **PERx** (Pull Enable Register) kann dieser eingeschaltet werden.

3.1 Digital-Ein- und Ausgänge

Die meisten Ports haben alternative Funktionen, z.B. können die Ports A und B entweder als Digital-Ein-/Ausgänge *oder* als externer Multiplex-Adress-Datenbus genutzt werden. Nach dem Reset sind alle Ports als Digital-Eingänge konfiguriert. Sobald eine Alternativfunktion aktiviert wird, ist der entsprechende Anschluss nicht mehr als Digital-Ein-/Ausgang verfügbar:

Port x	*Datenregister/ Datenrichtung*	*Bemerkung/Einschränkungen*	*Interrupt-fähig*	*Alternative Funktion*
A	PORTA / DDRA	Pull-Up-Widerstände und Treiberstärke nur für den gesamten Port gleichzeitig einstellbar (Register PUCR statt PPSx und PERx, Register RDRIV statt RDRx)	Nein	Multiplex-Adress-Datenbus
B	PORTB / DDRB			
E	PORTE / DDRE	Wie PORTA; Pins 0 und 1 nur als Eingänge	Nein	Steuersignale für Adress-Datenbus
K	PORTK / DDRK	Wie PORTA		
H	PTH / DDRH		Ja	SPI-Schnittstelle
J	PTJ / DDRJ	Nur 4 bit (J.0, 1, 6, 7) verfügbar	Ja	CAN oder I²C
M	PTM / DDRM		Nein	CAN-Schnittstellen
P	PTP / DDRP		Ja	PWM-Ausgänge SPI-Schnittstellen
S	PTS / DDRS		Nein	Serielle Schnittstellen SCI und SPI
T	PTT / DDRT		Nein	Timer-Ein/Ausgänge

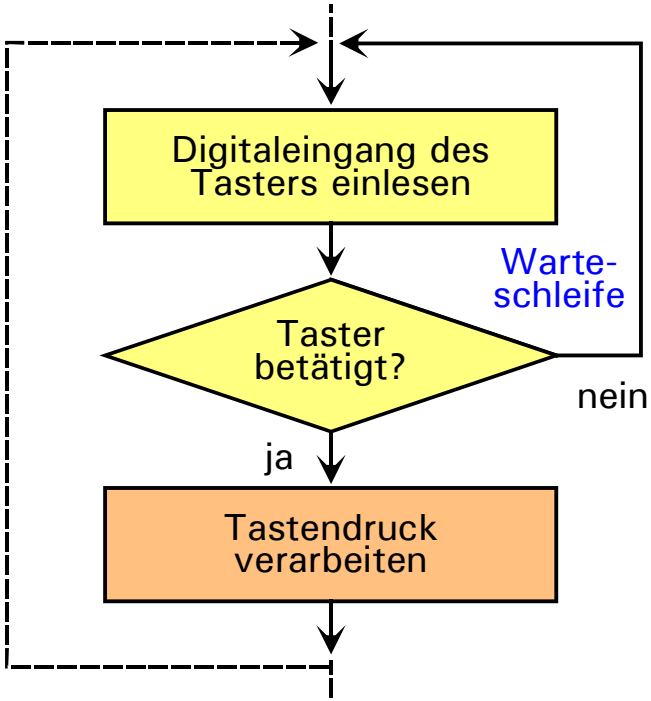
Die Ports H, J und P können flankengesteuert Interrupts erzeugen (siehe Kapitel 3.2).

Die Ports werden in der Regel nur beim Programmstart konfiguriert, anschliessend erfolgt der Zugriff ausschliesslich über das Datenregister (siehe Beispiel BlinkingLED in Kapitel 2.2)

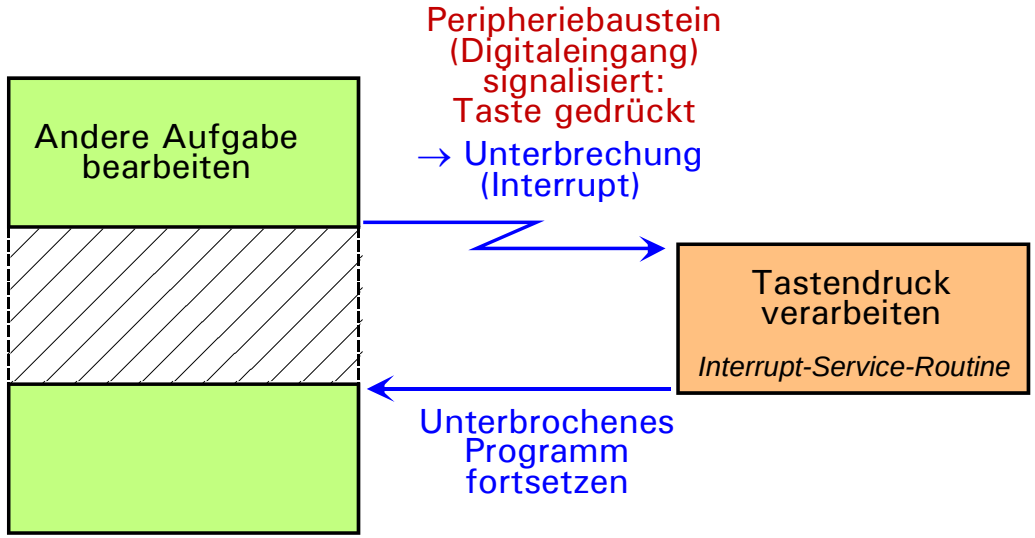
Verwendung der Ports auf dem Dragon12-Entwicklungsboard siehe Kapitel 2.1 und [3.11]

3.2 Interrupts (Unterbrechungen)

Aufgabenstellung: Synchronisation eines Programms mit einem externen Ereignis
z.B. Programm, das auf einen Tastendruck reagieren soll

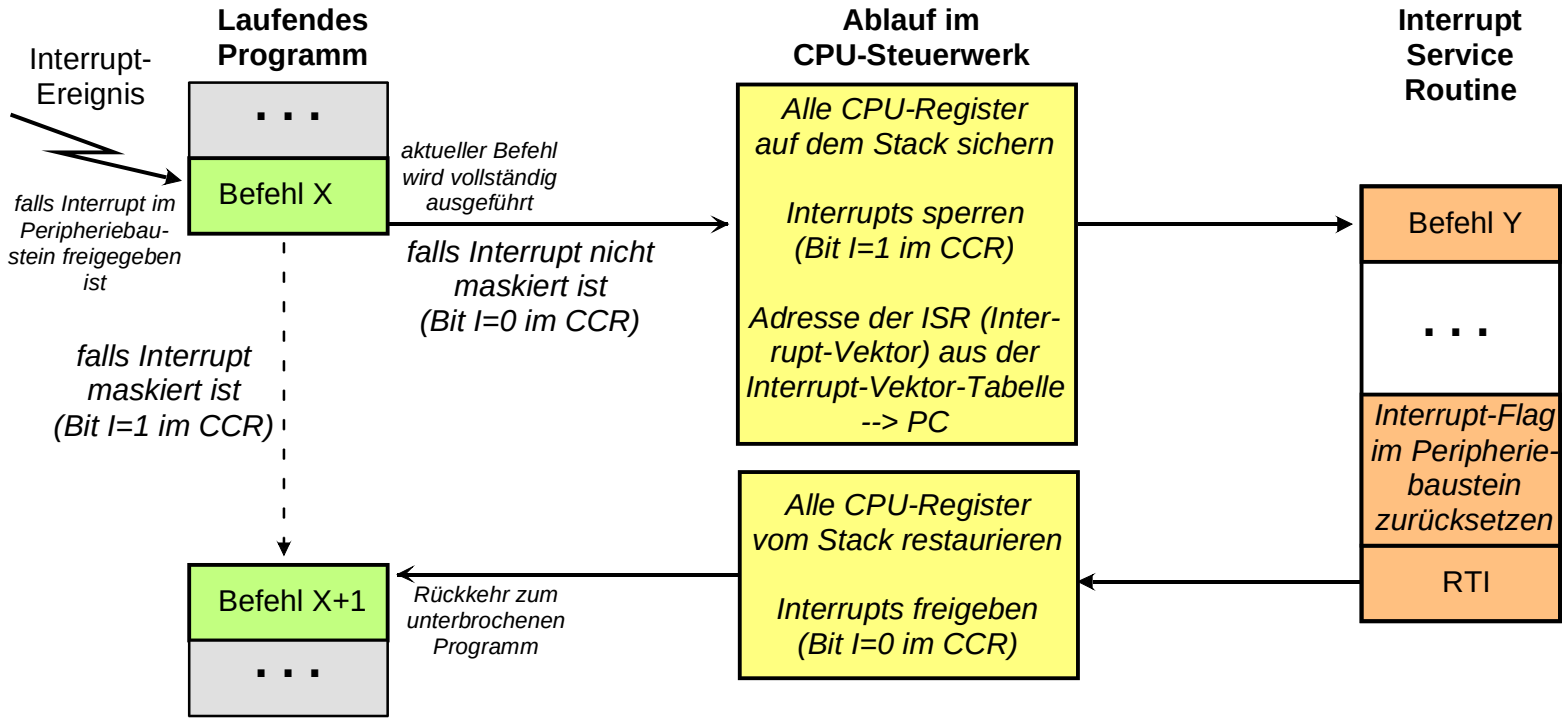


- Periodisches Abfragen (**Polling**) des Eingangs in einer Schleife bzw. bei Bedarf



- Bearbeitung einer anderen Aufgabe
- Schnittstellenbaustein signalisiert der CPU (Unterbrechung, **Interrupt**), wenn die Taste gedrückt wird
- CPU unterbricht andere Aufgabe, bearbeitet Unterbrechung (in einer **Interrupt-Service-Routine ISR**) und kehrt danach zur unterbrochenen Aufgabe zurück

Ablaufschema



Damit eine Interrupt-Service-Routine ISR ausgeführt wird,

- muss die Interrupt-Sperre (Interrupt Mask Bit I im CCR) der CPU inaktiv sein
- muss die Adresse des ersten Befehls der ISR in die Interrupt-Vektor-Tabelle (siehe nächste Seite) eingetragen werden
- muss der Peripheriebaustein so konfiguriert werden, dass ein Interrupt-Ereignis erzeugt wird (Interrupt Enable IE)
- muss die Interrupt-Signalisierung (Interrupt Flag IF) im Peripheriebaustein am Ende der ISR wieder zurückgesetzt werden.

Interrupt-Vektor-Tabelle (Auszug, vollständige Tabelle siehe [3.2])

Nr.	*Adresse*	*Zweck*	*Maskierbar*
0	$FFFE	Reset	Nein [*1]
. . .	. . .	. . .	. . .
2	$FFFA	COP: Watchdog-Interrupt (Computer Operating not Properly)	Ja
3	$FFF8	TRAP: Unimplemented Opcode	Nein [*1]
4	$FFF6	SWI: Software-Interrupt, aufgerufen durch Befehl SWI	Nein [*1]
5	$FFF4	XIRQ: Externes, nicht maskierbares Interrupt-Request-Signal	Nein [*2]
6	$FFF2	IRQ: Externes Interrupt-Request-Signal	Ja
7	$FFF0	RTI: Real Time Interrupt	Ja
. . .	. . .	. . .	. . .
8	$FFEE	Timer channel 0	Ja
. . .	. . .	. . .	. . .
15	$FFE0	Timer channel 7	Ja
. . .	. . .	. . .	. . .
20	$FFD6	SCI0: Erste serielle Schnittstelle	Ja
21	$FFD4	SCI1: Zweite serielle Schnittstelle	Ja
22	$FFD2	ATD0: AD-Umsetzer	Ja
. . .	. . .	. . .	. . .
25	$FFCC	PTH: Digitaleingänge Port H	Ja
. . .	. . .	. . .	. . .
127	. . .	. . .	. . .

[*1] Die „nicht-maskierbaren“ Interrupts sind unabhängig vom Interrupt-Mask-Bit I im CCR.
[*2] Der Eingang XIRQ kann durch das Mask-Bit X im CCR maskiert werden.

Auslöser von Interrupts

- Reset der CPU durch ein externes Signal, den Watchdog oder die Taktgenerator-Überwachung (im Unterschied zu echten Interrupts kehren diese ISRs üblicherweise nicht zum unterbrochenen Programm zurück sondern initialisieren die CPU neu)
- Hardware-Interrupts: Viele Peripheriebausteine können Interrupt-Ereignisse auslösen
- Software-Interrupt: Ausführen des Befehls SWI (wird vom Monitor-Programm verwendet)
- Ausnahmefehler (Exception oder Trap): Bestimmte Fehlersituationen, z.B. der Versuch, einen nicht definierten Opcode als Befehl zu dekodieren.

Priorisierung

- Falls mehrere Interrupt-Ereignisse gleichzeitig auftreten, wird diejenige Interrupt-Service-Routine zuerst ausgeführt, die in der Interrupt-Vektor-Tabelle die niedrigste Nummer hat, d.h. der Reset-Interrupt hat die höchste Priorität.
- Über das Register HPRIO kann die Priorität *eines* einzelnen Interrupts höher gesetzt werden als die Priorität aller anderen Interrupts
- Sollen während einer ISR andere Interrupts zugelassen sein, muss in der ISR die Interrupt-Sperre mit CLI aufgehoben werden.
- Interrupt-Ereignisse bleiben solange gespeichert, bis die Interrupt-Signalisierung im jeweiligen Peripheriebaustein zurückgesetzt wird.

Kommunikation mit Interrupt-Service-Routinen

- Interrupt-Service-Routinen werden asynchron zum übrigen Programm aufgerufen. Eine Übergabe von Parametern bzw. Rückgabewerte an/von der ISR sind nicht möglich (Ausnahme SWI). Kommunikation mit anderen Programmteilen nur über gemeinsame Variable möglich.

Beispiel in C: (CodeWarrior-Projekt **ButtonInterrupt.mcp**)

- Auslösen eines Interrupts bei Druck der Taste SW5 auf dem Dragon12-Entwicklungsboard (positive Flanke am Port H, Bit 0 der CPU)

```
. . .
void main(void)
{   EnableInterrupts;       //Allow interrupts
    . . .
    DDRB  = 0xFF;           //Port B.7...0 as outputs (LEDs)
    PORTB = 0x55;           //Turn on every second LED
    DDRH = 0x00;            //Configure port H.7…0 as inputs
    PPSH = 0x01;            //'1' trigger interrupt on positive slope on H.0
    PIEH = 0x01;            //'1' enables interrupt for input port H.0
    for(;;) { }             //Endless loop
}

interrupt 25 void ButtonISR(void) //ISR for interrupt 25 (port H interrupt)
{   PORTB = ~PORTB;         //Toggle LEDs on Port B
    PIFH = 0x01;            //'1' resets the interrupt flag
}
```

Wichtige Register für die 8 Digitaleingänge an Port H (siehe [3.3], Abschnitt 3.3.5):

DDRH ... Data Direction Register
PIEH ... Port Interrupt Enable Register (1=Freigabe, jeweils bitweise konfigurierbar)
PPSH ... Port Polarity Select Register (0=negative Flanke, 1=positive Flanke)
PIFH ... Port Interrupt Flag Register (1=Interrupt aufgetreten, Reset durch Schreiben von 1)

Beispiel in Assembler: (CodeWarrior-Projekt **ButtonInterruptAsm.mcp**)

- Aufgabenstellung wie oben

```
. . .
.vect:  SECTION              ; ROM: Interrupt vector section ---------------
        ORG   $FFCC
int25:  DC.W  isr25          ; Interruptvector for interrupt 25 (Port H)

.init:  SECTION              ; ROM: Code section ----------------------------
main:                        ; Begin of the program
Entry:  LDS  #__SEG_END_SSTACK  ; Initialize stack pointer
        CLI                  ; Enable interrupts

        . . .
        MOVB #$FF, DDRB      ; $FF -> DDRB:  Port B.7...0 as outputs (LEDs)
        MOVB #$55, PORTB     ; $55 -> PORTB: Turn on every other LED
        MOVB  #$00, DDRH     ; Configure port H.7...0 as inputs
        MOVB  #$01, PPSH     ; '1' trigger interrupt on positive slope
        MOVB  #$01, PIEH     ; '1' enables interrupt for input port H.0
loop:   BRA loop             ; Endless loop

; Interrupt Service Routine for interrupt 25
isr25:  COM  PORTB           ; Complement Port B: Toggle LEDs
        BSET PIFH, #1        ; Clear interrupt flag
        RTI                  ; Return from interrupt service routine (not RTS)
```

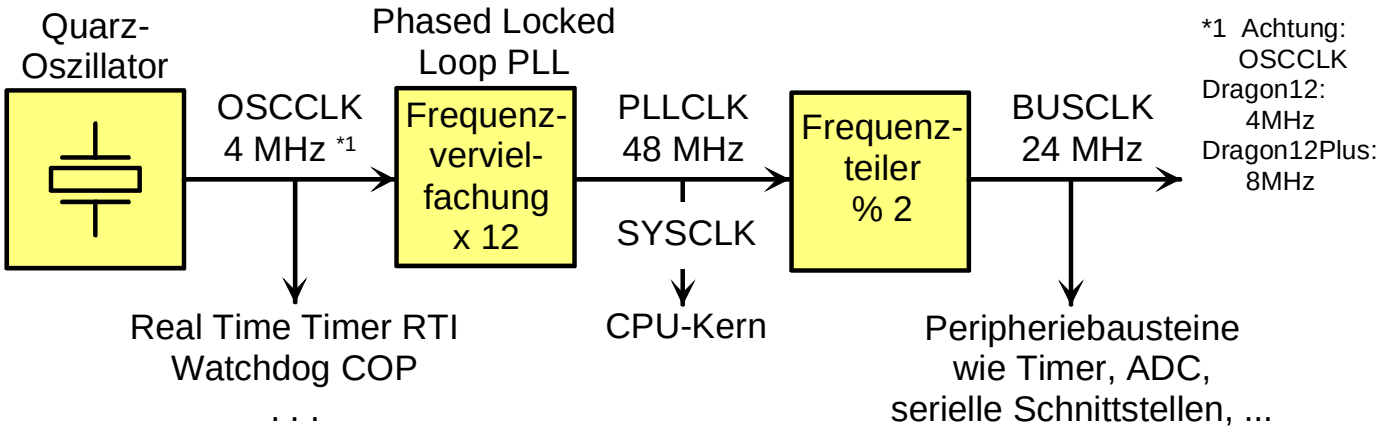
Prozessorzustand nach Reset

- PC wird mit dem Reset-Interrupt-Vektor geladen (auf dem Dragon12-Entwicklungsboard zeigt der Reset-Vektor auf das Monitor-Programm)
- I=X=S=1 im CCR-Register, d.h. Interrupts maskiert, Stop-Befehl nicht aktiviert
- andere Register, auch SP sind undefiniert
- Peripheriekomponenten sind abgeschaltet, u.a. die PLL des Taktgenerators, der Watchdog (COP) und der Realtime Timer (RTI)
- alle Digital-Ein-/Ausgänge sind als Eingänge geschaltet

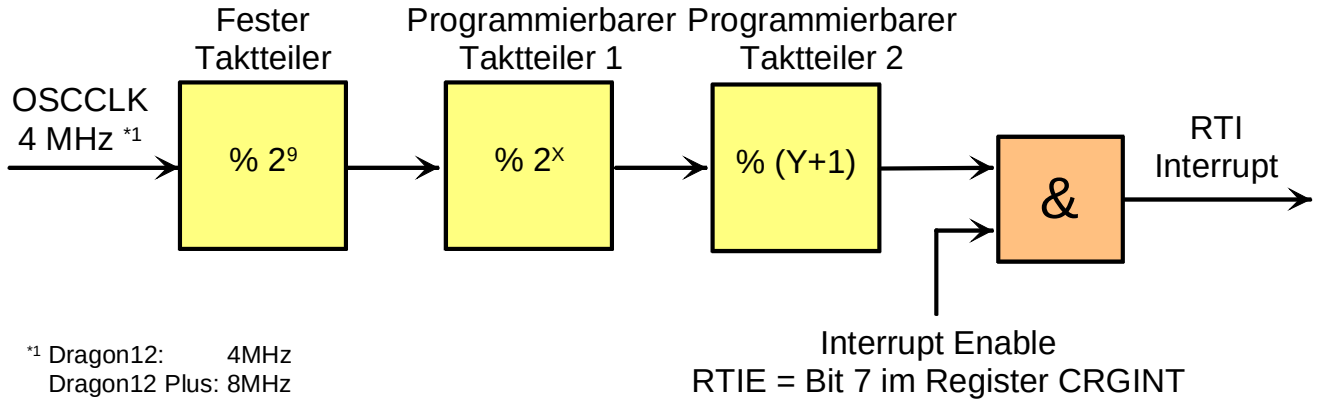
Taktgenerator und periodische Interrupts (Real Time Interrupts RTI)

- **Taktsignale** (siehe Clock & Reset Generator Module CRG [3.4] und Anhang)

  Der HCS12 verwendet einen komplexen, programmierbaren Taktgenerator, der auf dem Dragon12-Entwicklungsboard durch das Monitorprogramm folgendermassen konfiguriert wird:



- **Real Time Interrupt RTI** (siehe Clock & Reset Generator Module CRG [3.4])
  Am einfachsten lassen sich periodische Interrupts mit dem Real Time Interrupt erzeugen:



*1 Dragon12: 4MHz
Dragon12 Plus: 8MHz

Die Frequenz des Interrupts wird über die Teilerfaktoren X und Y eingestellt:

Register RTICTL

Bit	7	6	5	4	3	2	1	0
	0	X (3bit)			Y (4bit)			

$$f_{RTI} = \frac{f_{OSCCLK}}{2^{9+X} \cdot (Y+1)}$$

Y=0...15, X=1...7 (mit X = 000_B wird der Interruptgenerator vollständig abgeschaltet)

Beispiel: RTICTL = $7F_H$ →

Register CRGINT: Zur Freigabe des Interrupts (Real Time Interrupt Enable RTIE) muss Bit 7 im Register CRGINT auf 1 gesetzt werden, die übrigen Bits dürfen nicht geändert werden.

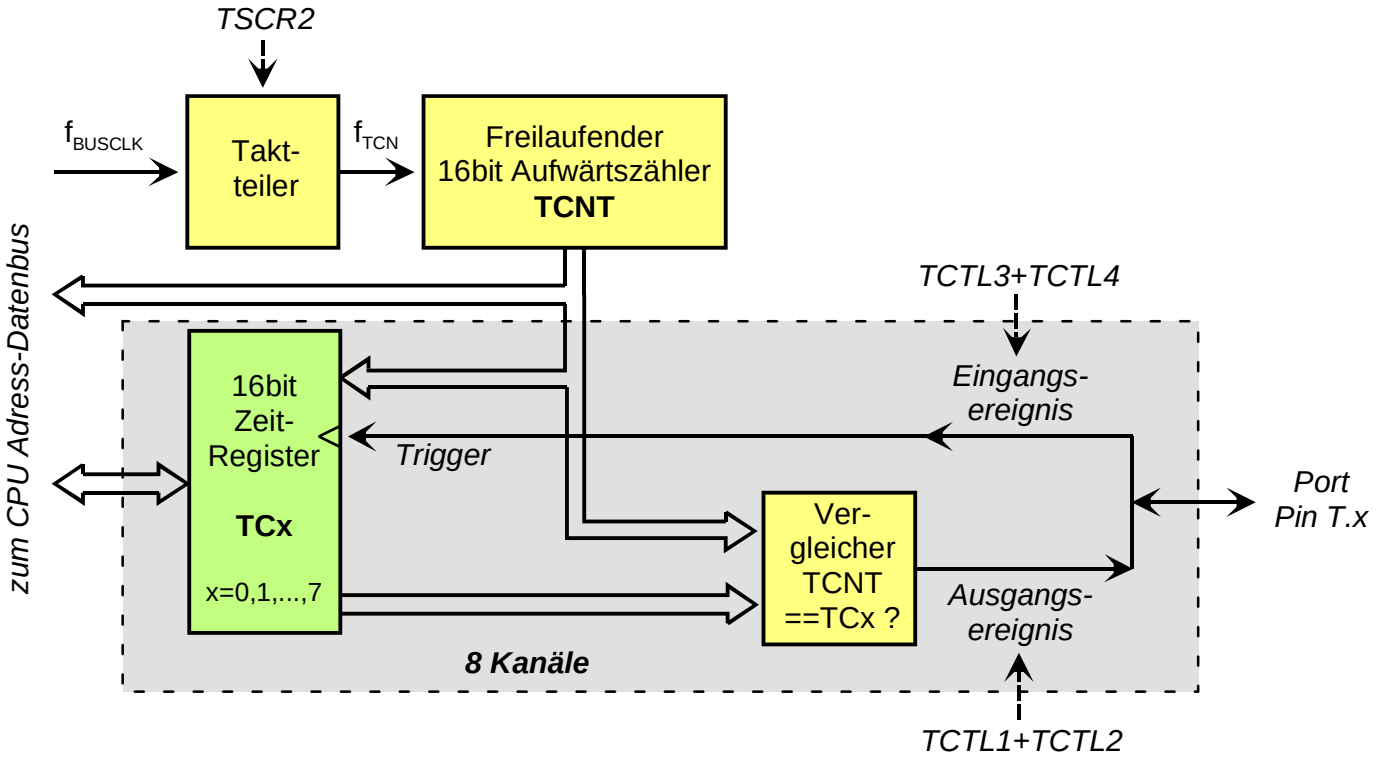
Register CRGFLG: Am Ende der zugehörigen Interrupt-Service-Routine muss die Interrupt-Signalisierung (RTI Interrupt Flag RTIF) Bit 7 im Register CRGFLG durch Schreiben einer 1 zurückgesetzt werden. Die übrigen Bits dürfen nicht geändert werden.

3.3 Timer-Einheit

(siehe Enhanced Capture Timer ECT [3.5])

Praktisch jeder Mikrocontroller verfügt über leistungsfähige Timer-Einheiten für Aufgaben wie

- Messung von Zeitdifferenzen in Programmen
- Messung von Zeitpunkten, Impulsdauern und Periodendauern von Eingangssignalen (**Input-Capture**-Betrieb): Abspeichern von TCNT bei Änderung eines Eingangssignals
- Erzeugung von Interrupts oder von Ausgangssignalen zu definierten Zeitpunkten (**Output-Compare**-Betrieb)



Der HCS12 auf dem Dragon12-Entwicklungsboard hat eine Timer-Einheit mit

- einem freilaufenden 16bit-Zähler TCNT
- 8 Kanälen, die mit dem Port T verbunden sind, die wahlweise als Input-Capture-Eingänge oder Output-Compare-Ausgänge verwendet werden können.

Der Enhanced-Capture-Timer ist ein relativ komplexes Modul, von dem hier nur die wesentlichsten Möglichkeiten dargestellt werden. Bei den Konfigurationsparametern werden hier nur diejenigen Werte angegeben, die von der Default-Konfiguration nach dem CPU-Reset abweichen, zu Einzelheiten siehe [3.5].

Konfiguration des freilaufenden 16bit Aufwärtszählers TCNT

- Steuerregister

TSCR1 (8bit Register)	Freigabe der Timer-Einheit Bit 7 = 1 Freigabe des Timer-Moduls Bit 6...0 = 0 Defaultwerte nach Reset
TSCR2 (8bit Register)	Einstellung des Zählertaktes Bit 7...3 = 0 Defaultwerte nach Reset Bit 2...0 Teilerfaktor x für Zählertakt Taktfrequenz des Zählers $f_{TCNT} = \frac{f_{BUSCLK}}{2^x}$ Beim Dragon12 ist f_{BUSCLK}=24MHz.

Der Zähler kann so konfiguriert werden (hier nicht dargestellt), dass er beim Überlauf einen Interrupt erzeugt. In der zugehörigen ISR kann dann ein Softwarezähler implementiert werden, so dass der Zähler beliebig erweitert wird.

Konfiguration der 8 Kanäle

- Steuerregister

TIOS (8bit Register)	Festlegung, ob ein Kanal als Input-Capture oder als Output-Compare arbeitet Bit y = 1 (y=0,1,…,7): Kanal y arbeitet als Output-Compare-Kanal Default: y=0, dh. Kanal als Input-Capture
TIE (8bit Register)	Interrupt Enable: Gibt Interrupts frei Bit y = 1 (y=0,1,…,7): Kanal y erzeugt einen Interrupt Default: y=0, dh. kein Interrupt Jeder Kanal hat seinen eigenen Interrupt, der ausgelöst wird, wenn ein entsprechendes Ein- oder Ausgangsereignis auftritt.
TFLG1 (8bit Register)	Interrupt Flag: Zeigt aufgetretene Interrupts an Bit y = 1 (y=0,1,…,7): Kanal y hat einen Interrupt ausgelöst Muss in der ISR durch Schreiben einer 1 in das entsprechende Bit zurückgesetzt werden.

- Zusätzliches Konfigurationsregister für den **Output-Compare-Betrieb**

Bit	7	6	5	4	3	2	1	0
TCTL1 (8bit)	Kanal 7		Kanal 6		Kanal 5		Kanal 4	
TCTL2 (8bit)	Kanal 3		Kanal 2		Kanal 1		Kanal 0	

Einstellung, welches Ausgangsereignis ausgelöst werden soll:
00 … Ausgang nicht aktiv (Anwendung: Interrupterzeugung, ohne dass Ausgangspin verwendet wird).
01 … invertiere Ausgang (toggle)
10 … Ausgang auf 0 (clear)
11 … Ausgang auf 1 (set)

- Zusätzliches Konfigurationsregister Register für den **Input-Capture-Betrieb**

	Bit 7	6	5	4	3	2	1	0
TCTL3 (8bit)	Kanal 7		Kanal 6		Kanal 5		Kanal 4	
TCTL4 (8bit)	Kanal 3		Kanal 2		Kanal 1		Kanal 0	

Einstellung, auf welche Eingangssignalflanke ein Kanal triggern soll (Eingangsereignis):
00 … Kanal nicht aktiv
01 … positive Flanke
10 … negative Flanke
11 … beide Flanken

- Zeitregister der einzelnen Kanäle
- Kanal 0 — TC0 (16bit Register)

. . .

. . .

- Kanal 7 — TC7 (16bit Register)

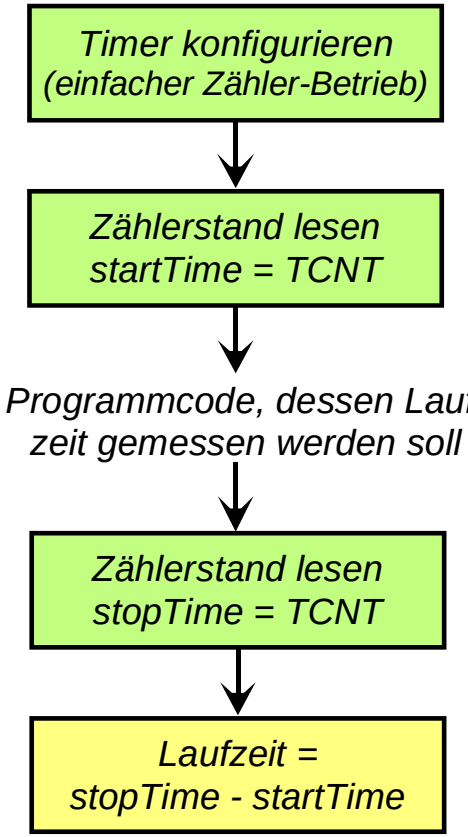
Im Output-Compare-Betrieb wird dieses Register geschrieben. Inhalt: Zeitpunkt (Zählerstand), bei dem das Ausgangsereignis des Kanals auslösen soll.

Im Input-Capture-Betrieb wird dieses Register gelesen. Inhalt: Zeitpunkt (Zählerstand), bei dem das Eingangsereignis aufgetreten ist.

Bei allen Software-Operationen, die das 16bit-Register TCNT verwenden, muss beachtet werden, dass TCNT sich sehr schnell ändert und periodisch „überläuft“:

- TCNT muss daher in einer atomaren 16bit-Operation gelesen werden. Würde man mit zwei 8bit-Lesebefehlen lesen, könnte sich TCNT zwischen den beiden Befehlen ändern.
- Verwendet man TCNT zur Messung von Zeitabständen, darf der Zeitabstand nicht größer sein als die Periodendauer von TCNT, in der Regel also $2^{16}/f_{TCNT}$. Bis zu diesem Zeitabstand sind Zählerüberlaufe wegen der mod 2^{16}-Arithmetik der ALU unproblematisch.

Anwendungsbeispiel: Laufzeitmessung in einem Programm (CodeWarrior-Projekt timer1.mcp)




```
.data: SECTION
startTime DS.W 1        ; TCNT at start of measurement
stopTime  DS.W 1        ; TCNT at end of measurement
runTime   DS.W 1        ; runTime= stopTime - startTime

.init: SECTION
main:
Entry:  . . .
; --- Initialize timer --------------------------------

; --- Program code to be measured ---------
    . . .

; --- Compute run time --------------------------------

    LDD  stopTime       ; Compute run time
    SUBD startTime
    STD  runTime        ; Einheit: Timer clock period
```

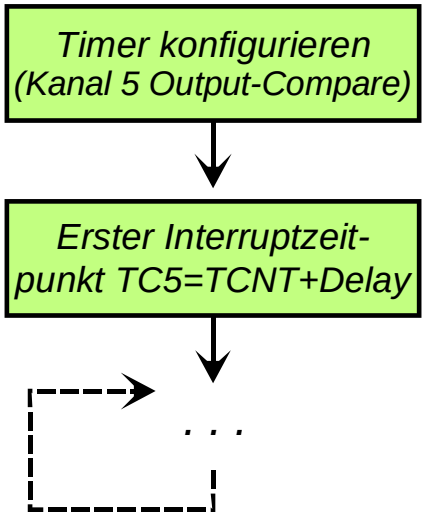
Hier: Auflösung der Messung: 1 Timer-Takt, d.h. 5,3µs Messbereich: 1 Timer-Periode, d.h. max. 350ms

Bessere Auflösung bei kleinererem Messbereich durch kleineren Teilerfaktor für den Zählertakt möglich

Anwendungsbeispiel: Ansteuerung des Dragon12-Piepsers im Output-Compare-Betrieb

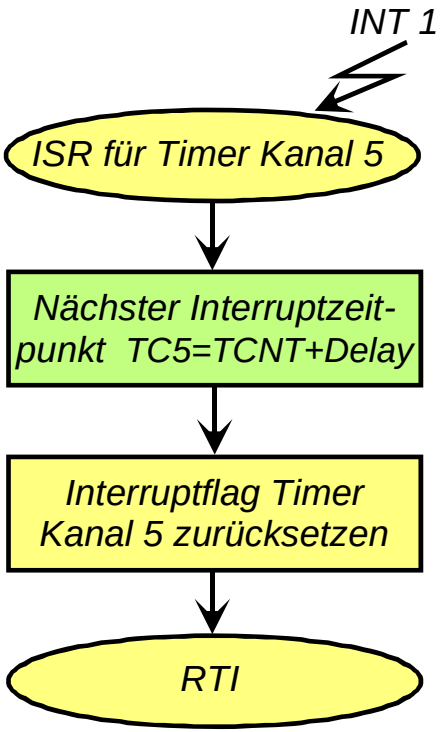
Der Piepser (Buzzer) an Port T.5 soll durch ein Rechtecksignal mit 500Hz und Tastverhältnis 1:1 angesteuert werden: (CodeWarrior-Projekt timer2.mcp)

Hauptprogramm:



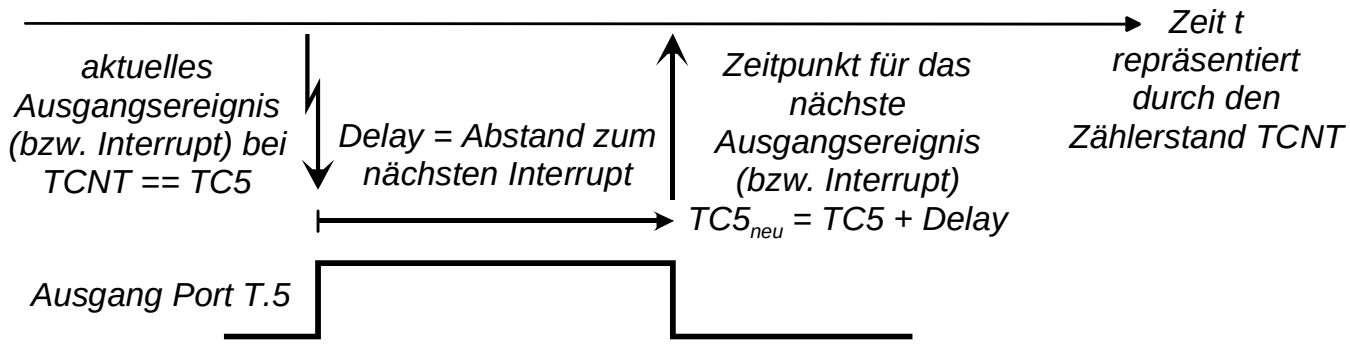

```
DELAY: EQU (24000/128)  ; Delay 1ms * 24Mhz / 2^7
. . .
.vect: SECTION          ; ROM: Interrupt vector entries
    ORG $FFE4
int13:  DC.W timer5Isr  ; timer channel 5 interrupt

.init: SECTION
main:. . .
; --- Initialize timer -------------------------------
    BSET TSCR1, #$80    ; Enable timer module
    MOVB #$07, TSCR2    ; Timer clock period 2^7/ 24MHz




lp: BRA  lp             ; Infinite loop
```

ISR:




```
; --- Interrupt service routine for timer channel 5
; Output port T.5 is toggled automatically, whenever
; this ISR is called

timer5Isr:

        RTI
```

"Zeitangaben" als ganzzahliges Vielfaches der Taktperiode des Zählers TCNT



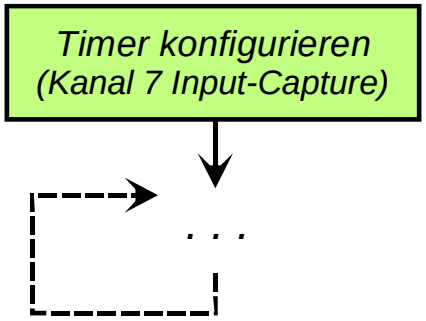
Setzt man die entsprechenden Bits in TCTL1 auf 0, so kann man mit einem Output-Compare-Kanal auch einfach eine ISR zu einem definierten Zeitpunkt aufrufen, ohne dass ein Ausgangssignal an einem Anschlusspin erzeugt wird.

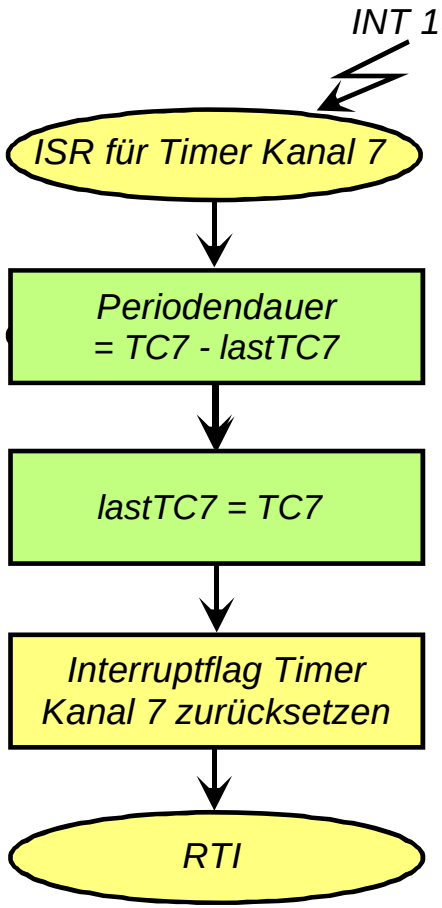
Anwendungsbeispiel: Messung der Periodendauer eines Signals im Input-Capture-Betrieb

Das Eingangssignal ist an Port T.7 angeschlossen und soll eine Frequenz im Bereich zwischen 10 Hz ... 10kHz haben. (CodeWarrior-Projekt timer3.mcp)

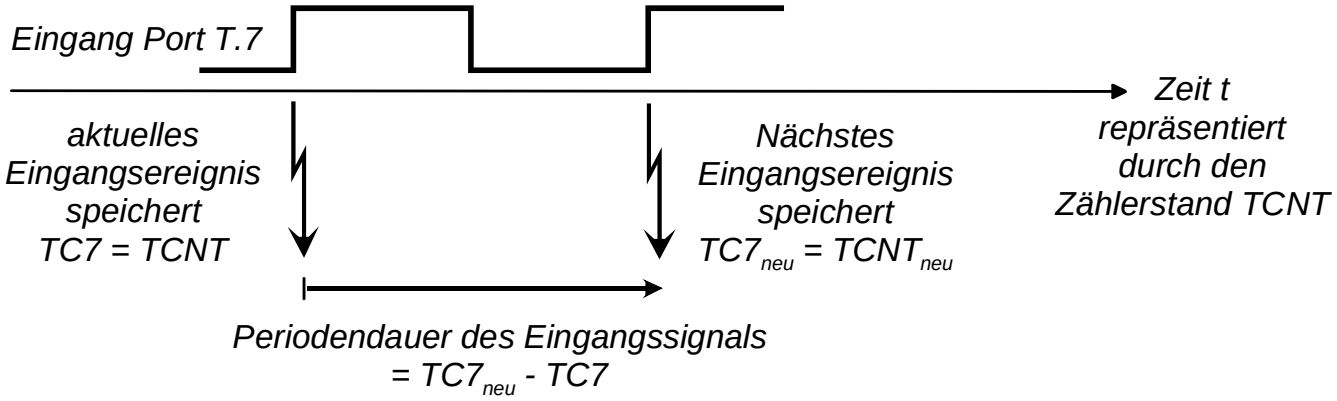
```
.data: SECTION
signalPeriod: DS.W 1    ; Signal period in TCN clock
                        ; periods
lastTC7:      DS.W 1    ; TC7 at last input event

.vect: SECTION          ; ROM: Interrupt vector entries
    ORG $FFE0
int15:  DC.W timer7Isr  ; Timer channel 7 interrupt

.init: SECTION
main: . . .
; --- Initialize timer -------------------------------
    BSET TSCR1, #$80    ; Enable timer module
    MOVB #$07, TSCR2    ; Timer clock period 2^7/ 24MHz



loop:   BRA  loop       ; Infinite loop
```

Hauptprogramm:






```
; --- Interrupt service routine for timer channel 7 ---
timer7Isr:
    LDD TC7             ; Get current TC7
    PSHD

    SUBD lastTC7        ; Compute signalPeriod
    STD  signalPeriod   ;   = current TC7 - last TC7

    PULD                ; Save current TC7
    STD  lastTC7        ;   for next measurement

    BSET TFLG1, #$80    ; Reset interrupt flag of ch. 7

    RTI
```

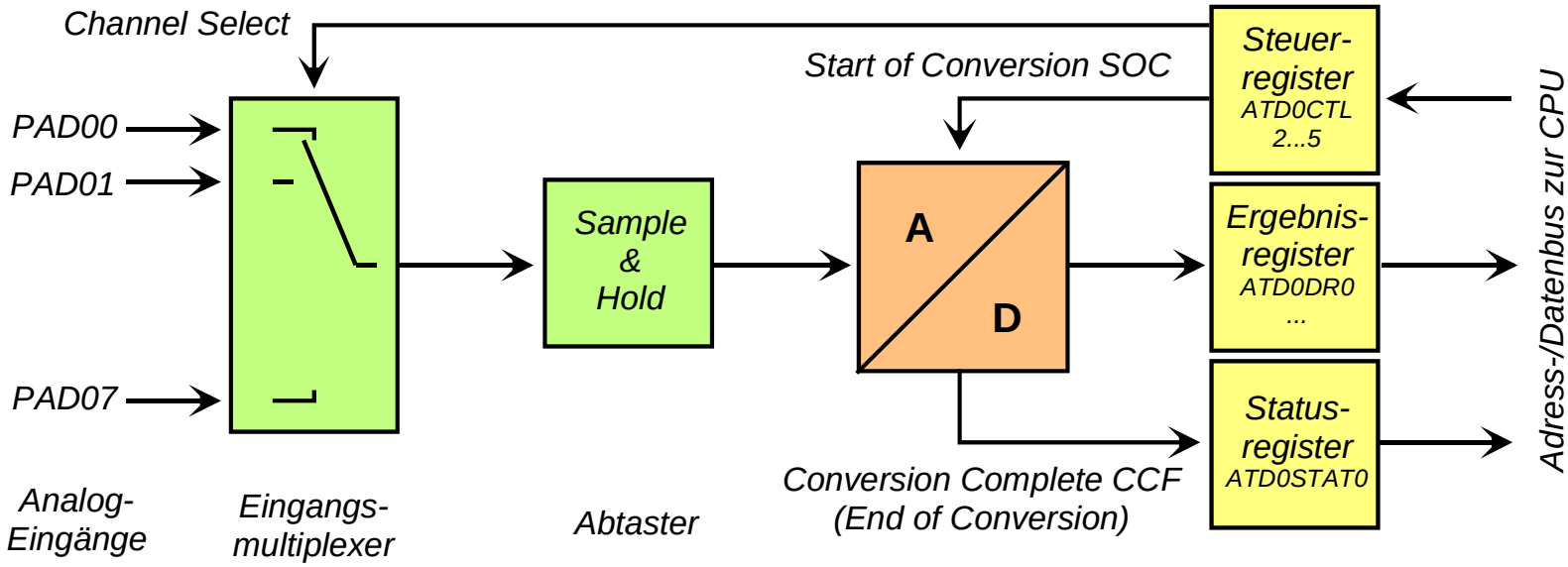


"Zeitangaben" als ganzzahliges Vielfaches der Taktperiode des Zählers TCNT

3.4 Analog-Digital-Umsetzung

(siehe Analog To Digital converter ATD [3.8])

- Der auf dem Dragon12-Entwicklungboard eingebaute HCS12-Mikrocontroller besitzt **zwei unabhängige 10bit-Analog-Digital-Umsetzer ATD0 und ATD1**.
- Jeder Wandler arbeitet nach dem Prinzip der **sukzessiven Approximation** und verfügt über ein **vorgeschaltetes Sample & Hold-Glied**.
- Die **Wandlungsdauer** beträgt 14 Takte (2 Takte für das Umschalten des Eingangsmultiplexers + 2 Takte für den Sample-Vorgang + 10 Takte für die 10bit A/D-Umsetzung), bei 2MHz Wandlertakt also **7µs**.
- Über einen vorgeschalteten **Eingangsmultiplexer** kann jeder Wandler einen von **8 Eingangskanälen je Wandler** messen: ATD0: Port PAD.07 ... 00, ATD1: Port PAD.15...08.



Die beiden Wandler ATD0 und ATD1 sind gleich aufgebaut. Im folgenden werden nur die grundsätzlichen Möglichkeiten dargestellt, zu spezielleren Betriebsarten siehe [3.8].

Die einmalig erforderliche Konfiguration des ATD-Moduls erfolgt über die Steuerregister:

- Steuerregister

(alle Register sind 8bit breit, wenn nicht anders angegeben)

ATD0CTL2	**Freigabe des ADC und Interrupt-Steuerung** Bit 7 = 1 Freigabe (Einschalten) des ADC-Moduls Bit 6 = 1 Automatisches Rücksetzen des CCF-Flags beim Lesen des Ergebnisregisters Bit 5 ... 2=0000_B Diverse Optionen, nicht ändern Bit 1 = 1 Interrupt Enable (nach Wandlungsende) Bit 0 = 1 Interrupt Flag, zeigt an dass Interrupt aufgetreten ist, Zurücksetzen durch Schreiben einer ‚1'
ATD0CTL3	**Wandlungssequenz** Bit 7=0 Default Bit 6 ... 3 Sequence Count SC, siehe unten Bit 2 ... 0 = 000_B Default
ATD0CTL4	**Auflösung und Wandlungsdauer** Bit 7 = 0 10bit Auflösung (Bit 7 = 1 ... 8bit Auflösung) Bit 6,5=00_B Abtastdauer des Sample & Hold-Glieds 2 Takte Bit 4..0=00101_B Teilerfaktor für Wandlertakt f_{ADC}=2MHz bei f_{BUSCLK}=24MHz
ATD0CTL5	**Datenformat und Start einer Wandlung** Bit 7...5 = 100_B Ergebnisse im Ergebnisregister rechtsbündig ohne Vorzeichen, d.h. 0V = 0_D, 5V = 1023_D Softwaretriggerung, kein freilaufender Betrieb Bit 4 Multichannel conversion MULT, siehe unten Bit 3 = 0 Default Bit 2 ... 0 Channel select Code C, siehe unten

Ob die Wandlung beendet und das Wandlungsergebnis im Ergebnisregister auslesbar ist, kann im Polling-Betrieb über das Statusregister abgefragt werden:

• Statusregister	ATD0STAT0	Bit 7 = 1	Wandlung beendet
		Bit 6 … 0	Diverse, seltener benötigte Informationen

Das Wandlungsergebnis steht im 16bit-**Ergebnisregister ATD0DR0** (oder in einem der daran anschliessenden Register ATD0DR1, …, ATD0DR7, siehe unten).

Betriebsarten und Start der A/D-Umsetzung:

A. Einmalige Messung eines einzelnen Kanals

Dazu stellt man ein	in ATD0CTL3$_{6...3}$:	SC=0001$_B$ einzelne Messung
	in ATD0CTL5$_4$:	MULT=0 kein Mehrkanalbetrieb
	in ATD0CTL5$_{2..0}$:	C = zu messender Eingangskanal 0, 1, …, 7
Wandlungsstart	erfolgt, sobald ATD0CTL5 geschrieben wird	
Wandlungsende	ATD0STAT07=1 (Abfrage durch Polling) bzw. Aufruf der ISR	
Wandlungsergebnis	in ATD0DR0 (unabhängig vom Kanal)	

B. Mehrfache Messung eines einzelnen Kanals

Einstellung wie A., außer	in ATD0CTL3$_{6...3}$:	SC= Anzahl der Messungen 1, 2, …, 8
Wandlungsergebnis:	in ATD0DR0, ATD0DR1, … (erster Messwert, zweiter Messwert usw.)	

C. Einmalige Messung mehrerer aufeinanderfolgender Kanäle

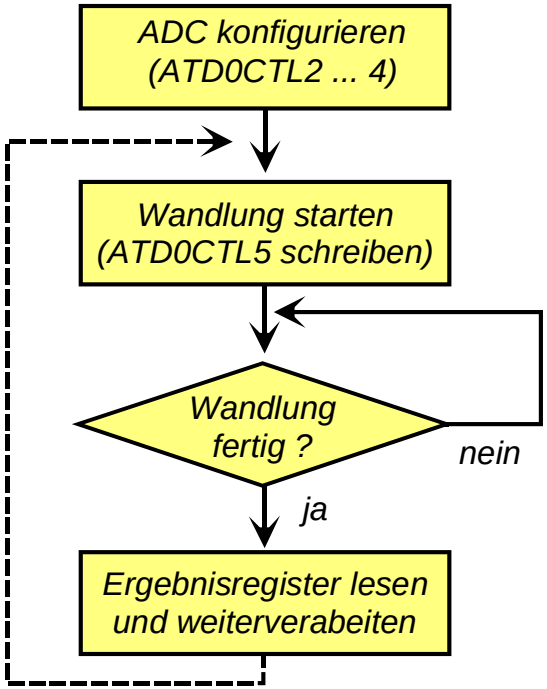
Einstellung wie A., außer in ATD0CTL3$_{6...3}$: SC= Anzahl Kanäle 1, 2, ..., 8

in ATD0CTL5$_4$: MULT=1 Mehrkanalbetrieb

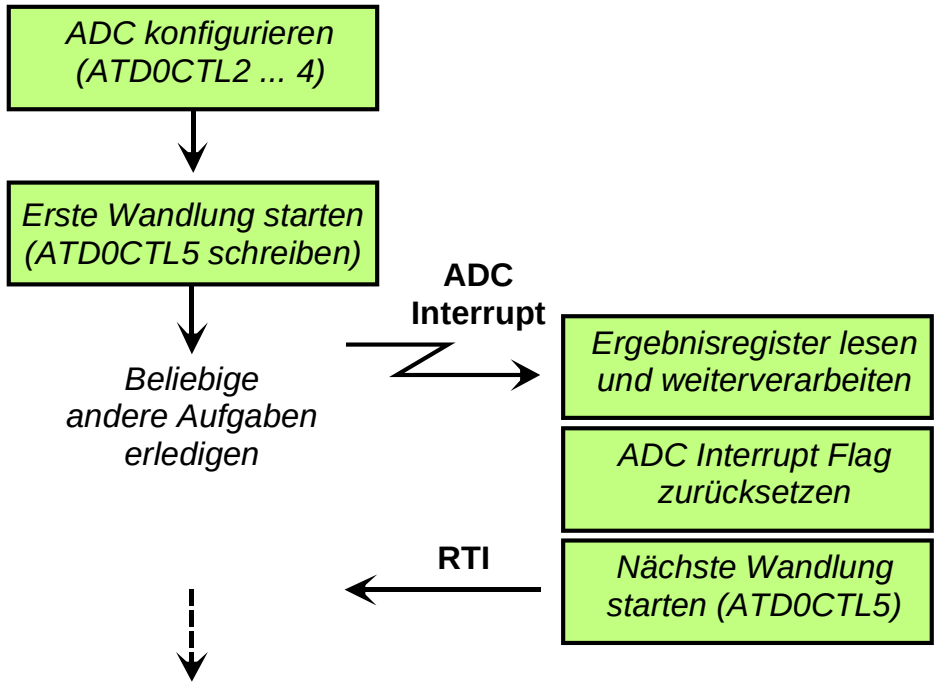
in ATD0CTL5$_{2...0}$: C = erster zu messender Kanal 0, 1, ..., 7

Startet man bei C=6 mit SC=4, werden die Kanäle 6, 7, 0 und 1 gemessen.

Wandlungsergebnis: in ATD0DR0, ATD0DR1, ... (erster gemessener Kanal, zweiter Kanal usw.)



Polling-Betrieb



Interrupt-Betrieb

Beispiel: (CodeWarrior-Projekt **ADCInterrupt.mcp**)

- Messung des Kanals 7 (Potentiometer) auf dem Dragon12-Entwicklungsboard
- Ausgabe des Mittelwertes von 4 Messungen in binärer Form auf die LED-Zeile

```
. . .
.data:  SECTION                 ; RAM: Variable data section
value:  DS.W  1                 ; Measurement value 10bit, right justified

.vect:  SECTION                 ; ROM: Interrupt vector entries



.init: SECTION                  ; ROM: Code section
main:
Entry:  . . .
    ; --- Initialize ATD0 ----------------------------------------------



lp: LDD  value                  ; Show upper 8bits of meas. value on LEDs 7...0
    LSRD
    LSRD
    STAB PORTB
    BRA  lp
```

```
    ; --- Interrupt Service Routine for interrupt 22 ----------------------
adcIsr:
```

Achtung: Die **Ergebnisregister** ATD0DR0, ... müssen ge**lesen** werden (um das CCF-Flag automatisch zurückzusetzen), **bevor** die **nächste Wandlung gestartet wird.**

Statt die Analog-Digital-Umsetzung durch die Software zu triggern (Schreiben in ATD0CTL5), gibt es zwei weitere, hier nicht näher beschriebene Formen der Triggerung:

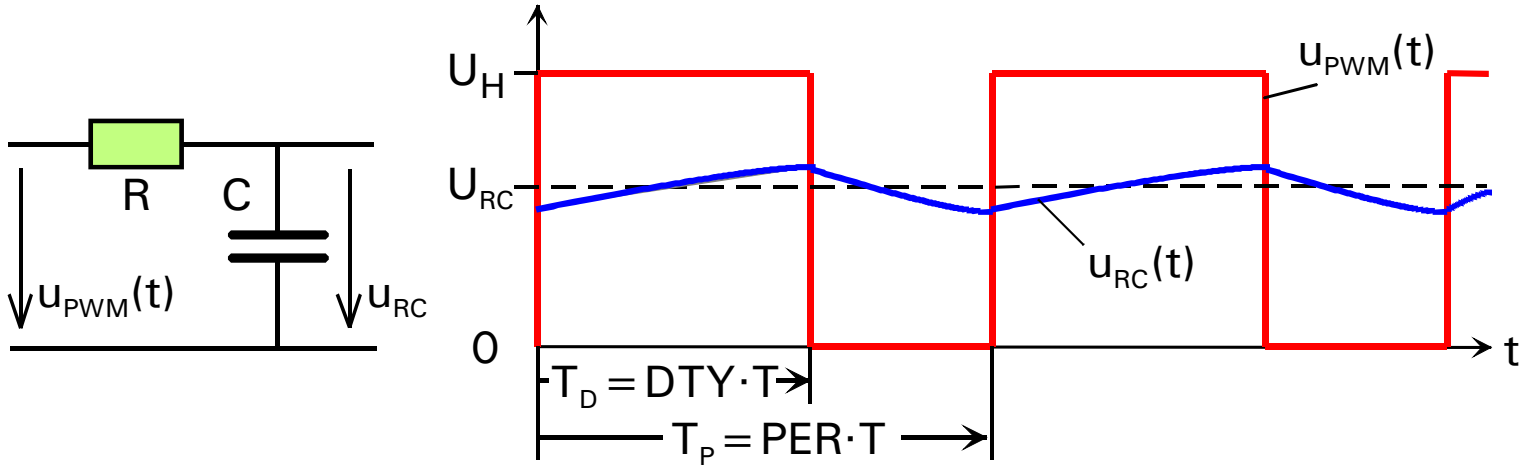
- Automatische Triggerung: In diesem Modus (Scan-Modus oder freilaufender Betrieb, wird mit Bit 5=1 im Register ATD0CTL5 aktiviert) wird nur die allererste Wandlung durch die Software gestartet, danach startet der ADC nach Ende jeder Umsetzung automatisch die nächste Umsetzung. Dadurch steht zwar ständig ein Messwert bereit, ohne dass die Software auf das jeweilige Wandlungsende warten muss, doch sind die Abtastzeitpunkte nicht mehr mit der Software synchronisiert.
- Hardware-Triggerung: Dabei wird die AD-Umsetzung statt durch Software durch ein externes Hardwaresignal, eine steigende oder fallende Signalflanke an Kanal 7 des ADC, getriggert.

3.5 PWM-Ausgänge

(siehe Pulsweitenmodulierte Impulssignale PWM [3.7])

Aufgabe:

- Erzeugung von Rechtecksignalen mit variabler Periodendauer T_P und Einschaltdauer T_D
  (D … Duty Cycle)



- Ausgabe quasi-analoger Information: Arithmetischer Mittelwert $U_{PWM} = U_H \frac{T_D}{T_P}$

- Nach Tiefpassfilterung, z.B. durch RC-Glied oder RL-Glied (z.B. Spule eines Motors) Erzeugung eines „echten“ Analogsignals, d.h. einfache D/A-Umsetzung

  → Ansteuerung von Servomotoren, Ventilen, Lampen, …

- Ändert man T_D bzw. T_P im laufenden Betrieb, kann man das „Analogsignal“ modulieren (Pulsweiten- bzw. Pulsfrequenzmodulation)

Das PWM-Modul des HCS12 verfügt über 8 PWM-Ausgangskanäle (Port P.7…0), wobei T_D und T_P für jeden Kanal einzeln mit einer Auflösung von jeweils 8bit eingestellt werden können:

- Register für Kanal x
  (alle Register 8bit)

  (x=0,...,7)

PWMPERx	Periodendauer T_P als Vielfaches der Taktperiode T_x (Um die maximale Auflösung zu erhalben, wird man in der Regel PWMPER=255 wählen)
PWMDTYx	Dauer der Phase T_D als Vielfaches der Taktperiode T_x (Es muss immer sein PWMPERx > PWMDTYx)
PWMCNTx	Zählerregister des PWM-Kanals x (Kann vom Anwender auf 0 gesetzt werden, um das PWM-Signal neu zu starten, wird in der Regel aber nicht verwendet).

Dazu gibt es noch drei 8bit Steuerregister, in denen jedes Bit genau einen Kanal steuert:

- Gemeinsame Steuerregister (1bit je Kanal)

PWME	Freigabe (Enable): Wenn ein Bit auf 1 gesetzt wird, wird das Rechtecksignal des entsprechenden Kanals generiert. Andernfalls kann der Port als normaler Digital-Ein-/Ausgang verwendet werden. Die Freigabe sollte erst erfolgen, nachdem alle PWM-Ausgänge inkl. Taktteilern konfiguriert sind.
PWMPOL	Polarität: Wenn ein Bit auf 1 gesetzt wird, beginnt das Rechtecksignal des entsprechenden Kanals mit H, sonst mit L.

PWMCLK

Auswahl des Taktsignals $T_{A/B}$ bzw. $T_{SA/SB}$:
Wenn ein Bit auf 0 gesetzt wird, verwendet der entsprechende Kanal das normale Taktsignal $T_{A\,/\,B}$, bei 1 das Taktsignal $T_{SA\,/\,SB}$

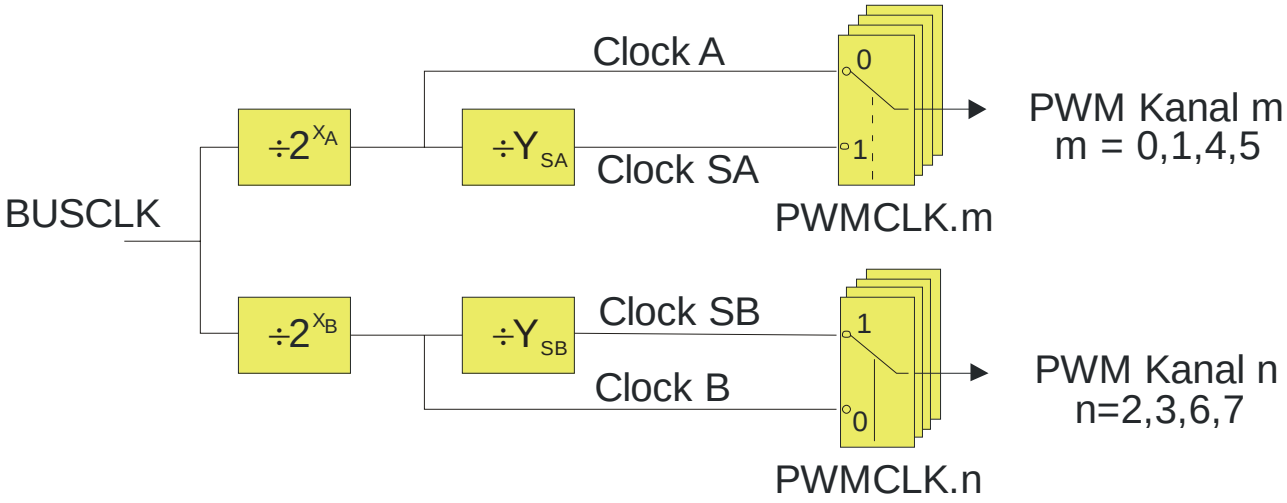
Innerhalb der PWM-Einheit sind insgesamt 4 Taktsignale vorhanden, von denen je Kanal jeweils 2 alternativ über PWMCLK ausgewählt werden können:

T_A oder T_{SA} für die PWM-Kanäle 0, 1, 4, 5 T_B oder T_{SB} für die PWM-Kanäle 2, 3, 6, 7

Die Taktsignale werden über Vorteiler aus dem Taktsignal BUSCLK erzeugt. Die Vorteiler sind über die Teilerfaktoren x_A bzw. x_B und y_{SA} bzw. y_{SB} programmierbar:

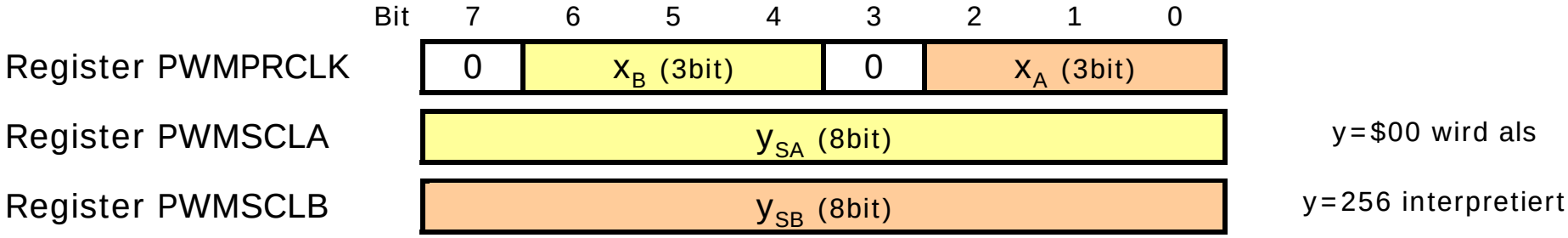
$$T_A = 2^{x_A} \cdot T_{BUSCLK} \qquad T_B = 2^{x_B} \cdot T_{BUSCLK}$$

$$T_{SA} = 2 \cdot y_{SA} \cdot T_A \qquad T_{SB} = 2 \cdot y_{SB} \cdot T_B$$



3.5 PWM-Ausgänge

Beim Dragon12 ist $T_{BUSCLK}=1/f_{BUSCLK}=1/24MHz$. Die Teilerfaktoren x_A, x_B, y_{SA} und y_{SB} werden über drei Register konfiguriert:



Die PWM-Einheit verfügt über einige weitere Spezialitäten und Konfigurationsregister, die hier nicht besprochen werden, weil sie nach dem Reset abgeschaltet sind und bei Nichtbenutzung nicht umkonfiguriert werden müssen.

Beispiel: Konfiguration für PWM Kanal 6 und 7 (CodeWarrior-Projekt **PWM1.mcp**)

```
MOVB #$80, PWMCLK

MOVB #$10, PWMPRCLK

MOVB #$02, PWMSCLB

MOVB #$C0, PWMPOL

MOVB #255, PWMPER6

MOVB #128, PWMDTY6

MOVB #255, PWMPER7

MOVB #32,  PWMDTY7

BSET PWME,#$C0
```

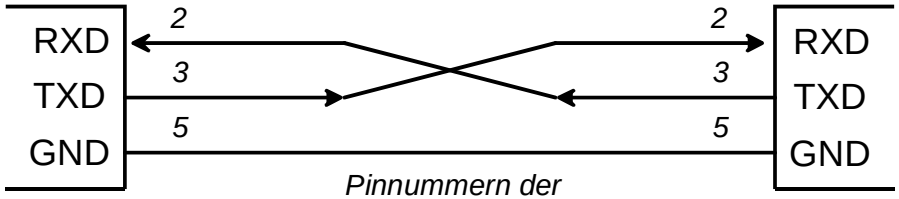
Aufgabe:

Geben Sie die maximal und die minimal mögliche Taktfrequenz des PWM-Rechtecksignals auf einem Dragon12-Entwicklungsboard an, wenn das Tastverhältnis T_D / T_P mit voller 8bit-Auflösung eingestellt werden soll.

3.6 Serielle Schnittstelle

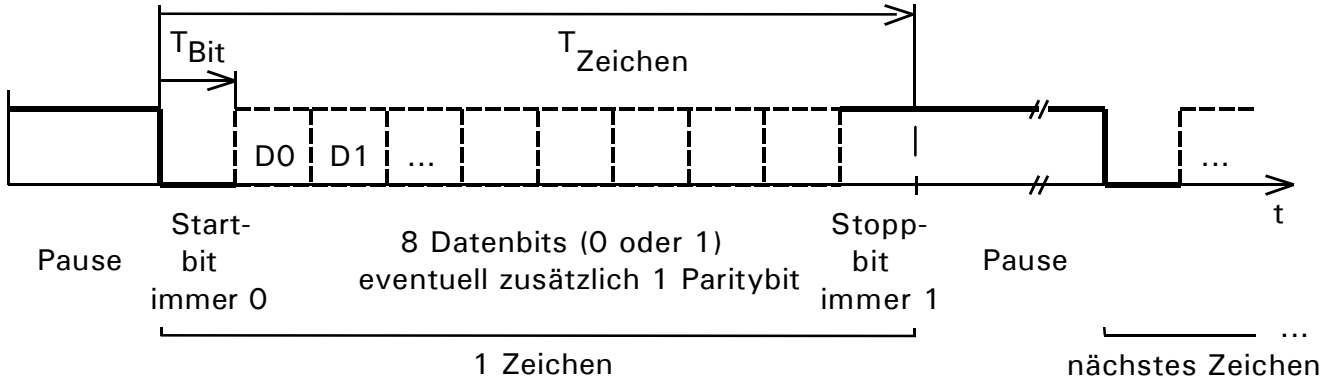
(siehe Serial Communications Interface SCI [3.6])

- Einfachste Art der bidirektionalen Voll-Duplex-Datenverbindung zwischen zwei Rechnern:



Pinnummern der Subminiatur D-9 Stecker/Buchsen

- Die Übertragung erfolgt byteweise („Zeichen“) und beginnt mit dem Startbit, einer ‚0’ auf der Leitung, da die Leitung bei einer Übertragungspause auf ‚1’ liegt.
- Dann folgen die 8 Datenbits, beginnend mit dem LSB.
- Optional wird ein Paritätsbit (gerade oder ungerade Parität der 8 Datenbits) übertragen.
- Die Übertragung wird durch ein Stoppbit abgeschlossen, das immer ‚1’ ist. Auf diesem Pegel bleibt die Leitung, bis das nächste Zeichen übertragen wird.

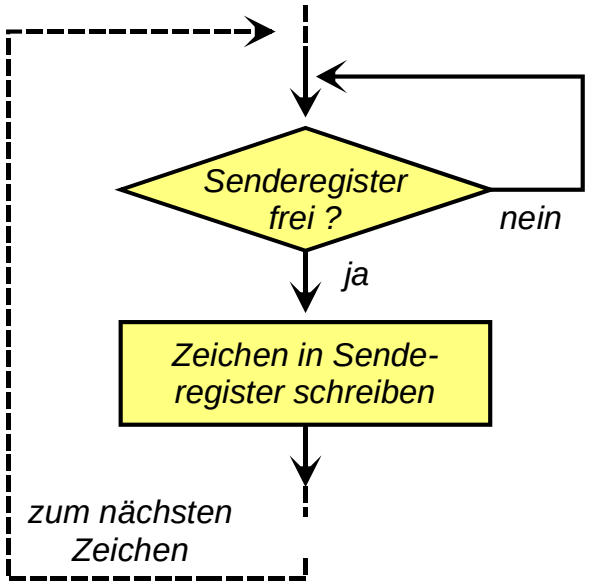


Baudrate = Bitrate

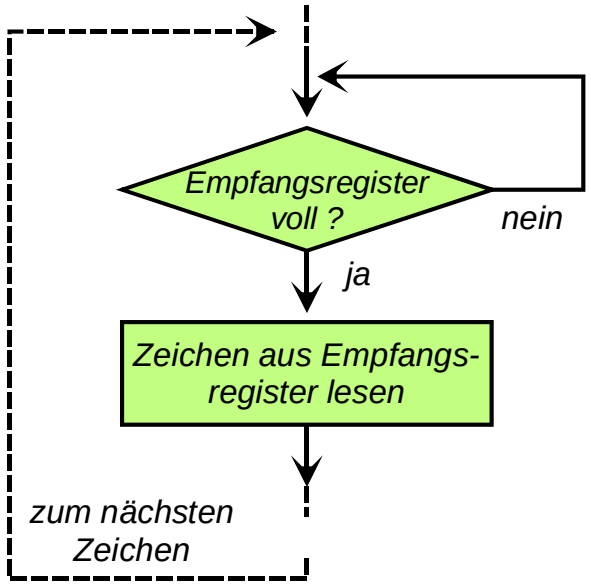
$$f_{bit} = \frac{1}{T_{bit}}$$

üblich 9.6 kbit/s, 19.2 kbit/s, 38,4 kbit/s, ..., 115.2 kbit/s

- Sender und Empfänger müssen mit derselben Bitrate und mit dem selben Zeichenformat (z.B. 8N1 = 8 Datenbits, keine (No) Parität, 1 Stoppbit) arbeiten. Da Sender und Empfänger nicht mit exakt demselben Taktsignal arbeiten, da dies nicht mit übertragen wird (asynchrone Datenübertragung), sind nur relativ niedrige Bitraten möglich.
- Die Übertragung eines einzelnen Zeichens wird durch eine Hardwarebaugruppe, den UART (Universal Asynchronous Receiver and Transmitter) automatisch ausgeführt, nachdem die Software ein Zeichen in das Senderegister des UARTs geschrieben hat. Nachdem ein Zeichen empfangen wurde, kann die Software es aus dem Empfangsregister des UART auslesen. Schematisch sieht der Vorgang im **Polling-Betrieb** folgendermassen aus:

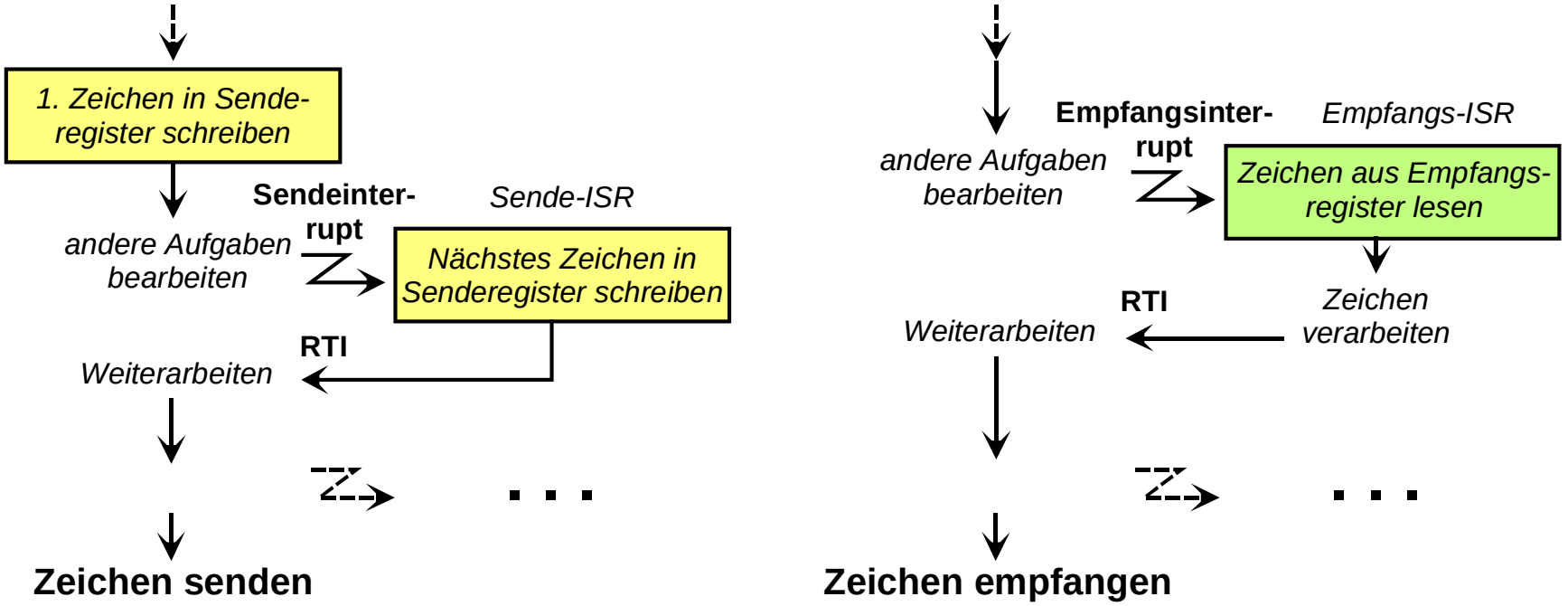


Zeichen senden



Zeichen empfangen

- Das Warten beim Senden, ob das Senderegister frei ist, d.h. ob das vorige Zeichen gesendet wurde, bevor das nächste Zeichen ins Senderegister geschrieben werden darf, sowie das Warten beim Empfang, bis ein Zeichen empfangen wurde, kann durch **Interruptbetrieb** vermieden werden. Dabei wird der UART so konfiguriert, dass er einen Interrupt erzeugt, wenn er ein Zeichen gesendet hat (**Sendeinterrupt**), bzw. wenn er ein neues Zeichen empfangen hat (**Empfangsinterrupt**). In der Praxis verwendet man oft die Mischform Polling beim Senden, Interruptbetrieb beim Empfangen.



Der HCS12 auf dem Dragon12-Entwicklungsboard hat zwei UARTs SCI0 und SCI1. Dabei wird SCI0 für die Kommunikation mit dem Debugger verwendet, SCI1 ist frei verfügbar und kann z.B. verwendet werden, um mit einem Terminalprogramm (Hyperterminal oder Terminal-Komponente des HCS12-Debuggers) auf dem PC zu kommunizieren.

Jede Schnittstelle hat drei Konfigurationsregister, die einmal gesetzt werden müssen:

• Baudrate Register (x=0 für SCI0, x=1 für SCI1)	SCIxBD (16bit Register!)	Teilerfaktor für die Takterzeugung $SCIxBD = \frac{f_{BUSCLK}}{16 \cdot f_{bit}}$ (beim Dragon12 ist f_{BUSCLK}=24MHz)
• Control Register 1	SCIxCR1 (8bit Register)	Default nach Reset: 8N1 (8 Datenbit, No Parity, 1 Stopbit) Falls Paritätsbit gewünscht wird: Bit 1=1 ... Sende/Empfange Paritätsbit Bit 0=1 ... Ungerade Parität (Odd) Bit 7...2 ... nicht verändern
• Control Register 2	SCIxCR2 (8bit Register)	Sender- und Empfängersteuerung Bit 2 = 1 ... Receiver Enable (Empfänger ein) Bit 3 = 1 ... Transmitter Enable (Sender ein) Bit 5 = 1 ... Receive Interrupt Enable Bit 7 = 1 ... Transmit Interrupt Enable Andere Bits = 0 Alle Interrupts verwenden denselben Interruptvektor. Wenn mehrere Interrupts aktiviert sind, muss in der ISR durch Abfrage des Statusregisters SCIxSR1 die Interruptursache ermittelt werden.

3.6 Serielle Schnittstelle

Sende- und Empfangs-Datenregister haben dieselbe Speicheradresse:

- Data Register

SCIxDRL (8bit Register)	Beim Schreiben: Senderegister Beim Lesen: Empfangsregister
SCIxDRH (8bit Register)	Datenregister obere 8bit, nur bei Betrieb mit >8 Datenbit nötig

Die Abfrage, ob das Senderegister frei bzw. das Empfangsregister voll ist, sowie andere Zustandsinformationen erfolgen über die beiden Statusregister

- Status Register 1

SCIxSR1 (8bit Register)	Bit 7 = 1 ... Senderegister frei Bit 5 = 1 ... Empfangsregister voll Die übrigen Bits zeigen verschiedene Fehlerbedingungen an, u.a. Bit 3 = 1 ... Empfangsregister durch neues Zeichen überschrieben bevor das vorige Zeichen ausgelesen wurde. Bit 0 = 1 ... Paritätsfehler

- Status Register 2

SCIxSR2 (8bit Register)	Bei üblichen Anwendungen nicht verwendet

Das Statusregister SCIxSR1 sollte immer, auch im Interruptbetrieb, vor dem Lesen oder Schreiben des Datenregisters gelesen werden, um die entsprechenden Flags zurückzusetzen.

Beispiel: Empfangen und Senden im Polling-Betrieb (CodeWarrior-Projekt **SerialPolling.mcp**)

```
SCIxBD: EQU  SCI1BD          ; On Dragon12 use SCI1, in the simulator use SCI0
SCIxCR1:EQU  SCI1CR1
CR:     EQU  $13
LF:     EQU  $10
. . .
.const:SECTION               ; ROM: Constant data
message1: DC.B  "Please enter a character", CR, LF, 0
. . .
.init: SECTION               ; ROM: Code section
main:
. . .                        ; Initialize the serial interface
    MOVW #13, SCIxBD         ; Set baud rate 115200 bit/sec
    MOVB #0,  SCIxCR1        ; Default format 8 data bits, no parity, 1 stop bit
    MOVB #$0C,SCIxCR2        ; Enable receiver and transmitter, no interrupts

    LDX  #message1           ; Send string to SCI
    JSR  puts
    JSR  getch               ; Get character from SCI, returns character in B
    JSR  putch               ; Send character in B to SCI
    . . .
```

Das Senden und Empfangen erfolgt in den drei Unterprogrammen **puts**, **getch** und **putch**.

```
getch:  ; --- Read a character from serial interface, return in B ------------
    BRCLR SCIxSR1, #$20, getch  ; Check 'Receive Data Flag' and wait
    LDAB SCIxDRL                ; Read received character
    RTS                         ; ... and return in B

putch:  ; --- Send a character in B to serial interface ---------------------
    BRCLR SCIxSR1, #$80, putch  ; Check 'Transmit Register Empty' and wait
    STAB SCIxDRL                ; Send character
    RTS                         ; ... and return

puts:   ; --- Send string to serial interface, X is a pointer to the string -
    TST  0, X                   ; Check for end of the string
    BEQ  done
    BRCLR SCIxSR1, #$80, puts   ; Check 'Transmit Register Empty' and wait
    MOVB 1, X+, SCIxDRL         ; Send one character
    BRA  puts                   ; Goto to the next character
done:RTS                        ; Send complete, return
```

Eine Version, die den Empfangsinterrupt statt Polling verwendet, finden Sie als CodeWarrior-Projekt **serialInterrupt.mcp** bei den Beispieldateien zu dieser Vorlesung.

Hinweis:
Wenn auf dem Dragon12-Entwicklungsboard die zweite serielle Schnittstelle SCI1 verwendet werden soll, muss Jumper J23 in Position RS232 umgesteckt werden.

Anhang: Taktgenerator

Falls das Dragon12-Entwicklungsboard mit einem im Flash-ROM gespeicherten Programm betrieben werden soll, muss der Taktgenerator der CPU initialisiert werden, da die CPU unmittelbar nach dem Reset nur mit einer niedrigen Taktfrequenz läuft. Normalerweise übernimmt das Monitor-Programm diese Aufgabe.

Zur Initialisierung des Taktgenerators kann z.B. folgende Befehlssequenz verwendet werden (siehe Clock & Reset Generator Module CRG [3.4]). Der Code setzt voraus, dass der Quarz mit f_{OSCCLK}=4MHz *1 arbeitet und stellt die PLL so ein, dass der Bustakt f_{BUSCLK}=24MHz wird:

```
        BCLR CLKSEL, #$80           ; Disconnect PLL from CPU (only in case)
        BSET PLLCTL, #$40           ; Turn on PLL
        MOVB #$05,SYNR              ; Set PLL multiplier
        MOVB #$00,REFDV             ; Set PLL divider     (beim Dragon12 Plus Board: #$01)
        NOP
        NOP
pllWait:BRCLR CRGFLG, #$08, pllWait ; Wait till PLL has locked
        BSET  CLKSEL, #$80          ; Connect PLL to CPU
         . . .
```

$$f_{PLLCLK} = 2\, f_{OSCCLK} \frac{SYNR+1}{REFDV+1}$$

max. 48MHz

$$f_{BUSCLK} = \frac{f_{PLLCLK}}{2}$$