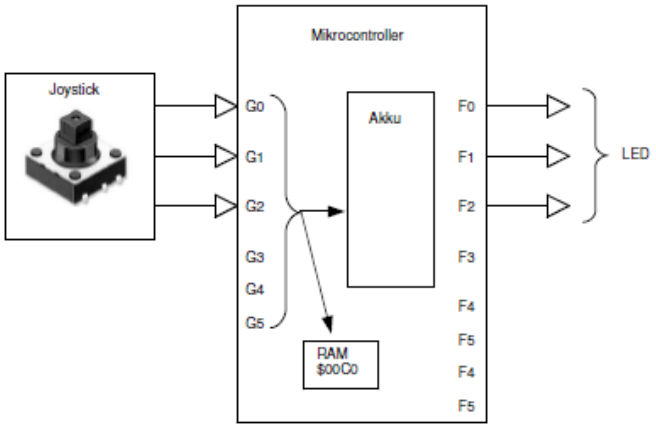
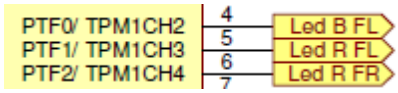
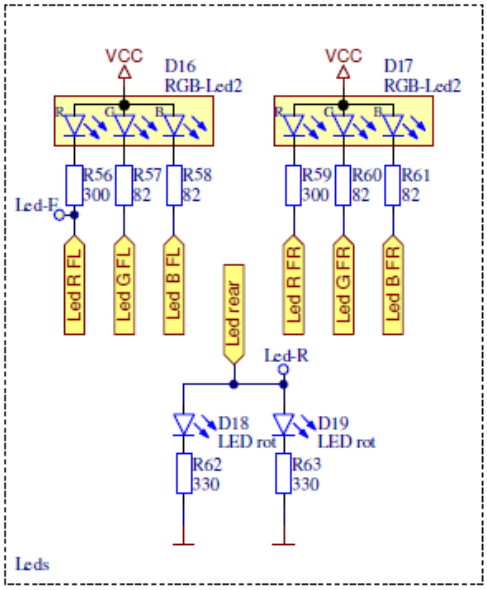
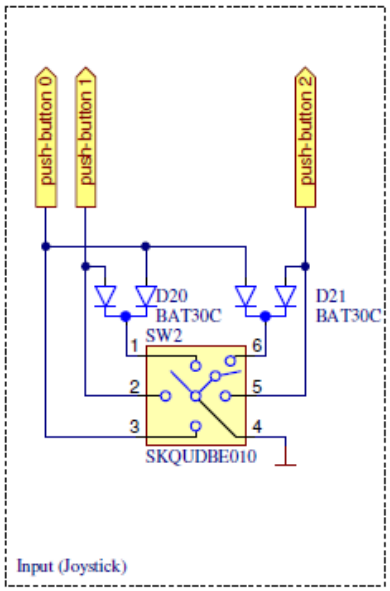
# Beschreibung der Grundaufgabe

Am Port G liege durch den Joystick 3-bit Wert an. Dieser Wert ist direkt auf Port G und in das interne RAM auf Adresse $00C0 zu kopieren. Jede Änderung des 3-bit Wertes soll nachgeführt werden. Füllen Sie Anschliessend die Wahrheitstabelle aus.




**MC9S08JM60CLH – Port / LED**

PTF0 Led B FL

PTF1 Led R FL

PTF2 Led R FR

| G2 | G1 | G0 | G2-0 Dez | SW Position | LED |
|---|---|---|---|---|---|
| 1 | 1 | 1 | 7 | Neutral | --- |
| 1 | 1 | 0 | 6 | Down | Blue FL |
| 1 | 0 | 1 | 5 | Left | Red FL |
| 1 | 0 | 0 | 4 | Up | Blue FL & Red FL |
| 0 | 1 | 1 | 3 | Right | Red FR |
| 0 | 1 | 0 | 2 | Press | Red FR & Blue FL |
| 0 | 0 | 1 | 1 | --- | Red FR & Red FL |
| 0 | 0 | 0 | 0 | --- | Red FR & Red FL & Blue FL |

# Assembler-Lösung

1. Ändern Sie ein auf dem Template „Ass_PortIO“ basierendes Assembler-Projekt so ab, dass es die gewünschte Funktion realisiert. Verwenden Sie dabei vorerst die symbolischen Namen der internen Steuerregister. Assemblieren Sie das File und kontrollieren Sie die Funktion auf dem MC CAR.

```
userMain: LDA   #$FF
          STA   PTGPE                   ; Port G Pull-Up Enable
          STA   PTFDD                   ; Port F DataDirection = Output

Loop:     LDA   PTGD                    ; Load Akku <-- Port G Data
          STA   PTFD                    ; Store Akku --> Port F Data
          BRA   Loop                    ; Branch --> Endlosschleife (Sprung zu Loop)
```

2. Ersetzen Sie im Programm jetzt die vordefinierten symbolischen Namen durch absolute Adressen (siehe „MC9S08JM60A.inc“ u. „HSLU_HCS08_Programing_Guide.pdf“ ) und prüfen Sie erneut die Funktion auf MC CAR.

```
userMain: LDA   #$FF
          STA   $00001858             ; PTGPE=$00001858 | Port G Pull-Up Enable
          STA   $0000000B             ; PTFDD=$0000000B | Port F DataDirection = Output

Loop:     LDA   $0000000C             ; PTGD=$0000000C | Load Akku <-- Port G Data
          STA   $0000000A             ; PTFD=$0000000A | Store Akku --> Port F Data
          BRA   Loop                  ; Branch --> Endlosschleife (Sprung zu Loop)
```

3. Überprüfen Sie jetzt die Funktion ihres Programmes im CW Debugger, in dem sie die Befehle **go**, **stepi**, **stop**, **mem**, **reg**, **bp** in der "Debugger Shell" verwenden.
   *In the CodeWarrior Eclipse IDE, the Debugger Shell is available from the C/C++ Perspective by choosing Window > Other > Debug > Debugger Shell, or from Debugger Perspective by choosing Window > Show View > Debugger Shell*

4. Im Memory Fenster des Debuggers kontrollieren Sie die transferierten Werte ( "Refresh while Running" wählen). Ab welcher Adresse beginnt Ihr Anwenderprogramm?
   *0x1969*

# C-Lösung

1. Führen Sie Aufgabe 1 in der Hochsprache C aus (Template „HCS08_C“). Überprüfen Sie im Debugger Disassembly-Fenster wie die C Anweisungen durch den Compiler übersetzt wurden. Was fällt Ihnen auf?
   *… mein Versuch:*

```
void main(void)
{
  EnableInterrupts;
  // user code
  PTGPE = 0xff; /* Port G Pull-Up Enable */
  PTFDD = 0xff; /* Port F DataDirection = Output */

  for(;;)
  {
    PTFD = PTGD; /* Port G (Taster) --> Port F (LED) */

    __RESET_WATCHDOG();     /* feeds the dog */
  } /* loop forever */

  /* please make sure that you never leave main */
}
```

# Assembler Übung aus Unterricht

| Location | Obj. Code | Source | | | Desc. |
|---|---|---|---|---|---|
| | --- | --- | Label10: EQU | $13 | Kein Obj. Code (nur Definition) |
| =0 | --- | --- | Konstanten: SECTION | | Definiert die Sektion |
| +1 | 0000 | 01 | Label1: DC | $01 | Hex 01 (Byte) in Flash speichern |
| +3 | 0001 | 02 03 04 | Label2: DC.B | $02, $03, $04 | Drei Konstante (Byte) speichern |
| +2 | 0004 | 12 34 | Label3: DC.W | $1234 | Ein Word (2 Byte) definieren |
| +4 | 0006 | 00 11 22 33 | Label4: DC.L | $00112233 | Ein Long (4 Byte) definieren |
| =0 | | --- | Daten: SECTION | | Neue Sektion definieren |
| +3 | 0000 | | Label5: DS | 3 | 3 Byte definieren |
| +4 | 0003 | | Label6: DS.B | 4 | 4 Byte definieren |

# Analyse Maschinencode

2. Versuchen Sie den nachfolgend gegebenen Maschinencode in mnemonischen Code umzusetzen (alle Byteangaben in Hex). Benutzen Sie hierzu die Vorlage "MC_Code.docx". Welche Bedeutung könnte die scheinbar unerklärbare Befehlsfolge in der Mitte des Programms haben?

```
$1960 -------- -------- --A60FB7 0BA680B7
$1970 09A6EFB7 C1B70A48 484848B7 08AEFFA6
$1980 FF4E0602 4E06024E 06024E06 025A26F1
$1990 AEFF4BED B6C149B7 C1B70A48 484848B7
$19A0 0820DA-- -------- -------- --------
```

| Start-Adresse (Hex) | | | | Assembler Befehl (Mnemonic) | OpCode (Hex) | | Operand1 (Hex) | | Operand2 (Hex) | | Kommentare |
|---|---|---|---|---|---|---|---|---|---|---|---|
| 1 | 9 | 6 | 9 | LDA #$0F | A | 6 | 0 | F | | | LDA IMM $0F \| 0F in Akku laden |
| 1 | 9 | 6 | B | STA $0B | B | 7 | 0 | B | | | STA DIR $0B \| Akku an Addr $0B speichern |
| 1 | 9 | 6 | D | LDA #$80 | A | 6 | 8 | 0 | | | $80 an Addr $09 speichern |
| 1 | 9 | 6 | F | STA $09 | B | 7 | 0 | 9 | | | |
| 1 | 9 | 7 | 1 | LDA #$EF | A | 6 | E | F | | | $EF an Addr $C1 speichern |
| 1 | 9 | 7 | 3 | STA $C1 | B | 7 | C | 1 | | | |
| 1 | 9 | 7 | 5 | STA $0A | B | 7 | 0 | A | | | $EF an Addr $0A speichern |
| 1 | 9 | 7 | 7 | LSLA | 4 | 8 | | | | | LSLA (Logical Shift Left) MSB → Carry |
| 1 | 9 | 7 | 8 | LSLA | 4 | 8 | | | | | 4x nach links schieben |
| 1 | 9 | 7 | 9 | LSLA | 4 | 8 | | | | | … |
| 1 | 9 | 7 | A | LSLA | 4 | 8 | | | | | Akku Inhalt danach: $F0 |
| 1 | 9 | 7 | B | STA $08 | B | 7 | 0 | 8 | | | Akku ($F0) an Addr $08 Speichern |
| 1 | 9 | 7 | D | LDX #$FF | A | E | F | F | | | LDX IMM $FF \| $FF in Register X Laden |
| 1 | 9 | 7 | F | LDA #$FF | A | 6 | F | F | | | FF in Akku laden |
| 1 | 9 | 8 | 1 | MOV $06 $02 | 4 | E | 0 | 6 | 0 | 2 | MOV DD (Dest) ← (Src) |
| 1 | 9 | 8 | 4 | MOV $06 $02 | 4 | E | 0 | 6 | 0 | 2 | 4x „sinnloser Code“ |
| 1 | 9 | 8 | 7 | MOV $06 $02 | 4 | E | 0 | 6 | 0 | 2 | … |
| 1 | 9 | 8 | A | MOV $06 $02 | 4 | E | 0 | 6 | 0 | 2 | → Delay |
| 1 | 9 | 8 | D | DECX | 5 | A | | | | | Register X Decrement (-1) \| Inhalt danach: $FE |
| 1 | 9 | 8 | E | BNE $F1 | 2 | 6 | F | 1 | | | BNE REL $F1 (zur akt. Addr + $F1) [signed]<br>$F1 = -128 + 113 = -15 → $0F<br>$90 + (-$0F) = $1981 |
| 1 | 9 | 9 | 0 | LDX #$FF | A | E | F | F | | | $FF in Register X Laden |
| 1 | 9 | 9 | 2 | DBNZA $ED | 4 | B | E | D | | | DBNZA INH → REL-Addr. !!!<br>(Decrement Akku and Branch if Not Zero)<br>$ED→ -19 → $1994 - $13 = $1981 |
| 1 | 9 | 9 | 4 | LDA $C1 | B | 6 | C | 1 | | | Wert von Addr. $C1 in Akku Laden \| $EF in Akku laden |
| 1 | 9 | 9 | 6 | ROLA | 4 | 9 | | | | | Rotate Left through Carry \| Akku Inhalt danach: $DE |
| 1 | 9 | 9 | 7 | STA $C1 | B | 7 | C | 1 | | | STA DIR $C1 \| $DE an Addr. $C1 speichern |
| 1 | 9 | 9 | 9 | STA $0A | B | 7 | 0 | A | | | STA DIR $0A \| $DE an Addr. $0A speichern |
| 1 | 9 | 9 | B | LSLA | 4 | 8 | | | | | LSLA (Logical Shift Left) |
| 1 | 9 | 9 | C | LSLA | 4 | 8 | | | | | 4x nach links schieben |
| 1 | 9 | 9 | D | LSLA | 4 | 8 | | | | | … |
| 1 | 9 | 9 | E | LSLA | 4 | 8 | | | | | Akku Inhalt danach: $E0 |
| 1 | 9 | 9 | F | STA $08 | B | 7 | 0 | 8 | | | $E0 an Addr. $08 speichern |
| 1 | 9 | A | 1 | BRA $DA | 2 | 0 | D | A | | | BRA $DA → REL-Addr. !<br>$DA = -128 + 90 = -38 → $26<br>$19A3 - $26 = $197D → Endlos Loop |

3. Geben Sie obigen Maschinencode direkt im Debugger Memory Browser ab der Adresse $1964 ein. Benutzen Sie die Debugger-Befehle „reg“ und „go“ um das Programm zu starten und überprüfen Sie Ihre vorausgesagte Funktion.
*done*

4. Erstellen Sie ein Assemblerprojekt, das die identische Funktion von 1.1 realisiert.

```
; Subroutinen
;*****************************************************************
Delay:      MACRO
            STA     TempA           ; Akku Inhalt sichern
            LDA     \1              ; Delay Wert als Makro-Parameter, EXT Adressierung
DelLoop:    DECA                    ; Akku dekrementieren bis 0
            BNE     DelLoop
            LDA     TempA           ; Akku Inhalt wieder herstellen
            ENDM

; Start des Benutzer-Codes
;*****************************************************************
userMain:   LDA     #$0F            ; Ports auf Ausgang setzen
            STA     PTFDD
            LDA     #$80
            STA     PTEDD
            LDA     #$EF            ; Bit auf Port F setzen (LEDs Low Active)
            STA     $00C1
            STA     PTFD
            LSLA                    ; Vier mal nach Links schieben
            LSLA
            LSLA
            LSLA
            STA     PTED            ; Auf Port E ausgeben (LED B FR)

Loop:       LDX     #$FF            ; Innerer Schleifenzaehler &
            LDA     #$FF            ; auesserer Schl.zaehler init

Iter:   IF UseMacro = 0
        ; Naive Lösung zum Zeit "verbraten"
            MOV     PTDD,PTBD       ; 4 x dummy Operation
            MOV     PTDD,PTBD
            MOV     PTDD,PTBD
            MOV     PTDD,PTBD
        ELSE
        ; (Etwas) elegantere Lösung zum Zeit verbraten, mit Macro
            Delay   DelVal
        ENDIF


            DECX                    ; X dekrementieren
            BNE     Iter            ; Springe zu Iter solange X != 0

            LDX     #$FF            ; Neu init. innere Schleife
            DBNZA   Iter            ; Dekrementiere A und springe zu Iter solange A > 0


            LDA     $00C1
            ROLA                    ; Bit schieben
            STA     $00C1
            STA     PTFD            ; Ausgabe neue Bitstellung
            LSLA                    ; Akku vier mal nach Links schieben
            LSLA
            LSLA
            LSLA
            STA     PTED            ; Auf Port E ausgeben (LED B FR)

            BRA     Loop            ; Ruecksprung


EndLoop:    BRA     *               ; Endlos-Loop (=Programmende)
```

# Arithmetik mit 8-Bit Zahlen

5. Studieren Sie die Befehle ADD und SUB im Freescale "HCS08 Family Reference Manual" und interpretieren Sie die beiden Folien „Carry und Overflow beim ADD/SUB-Befehl“.

**ADD** **Add without Carry** **ADD**

**Operation**

A ← (A) + (M)

**Description**

Adds the contents of M to the contents of A and places the result in A

**Condition Codes and Boolean Formulae**

| V | | | H | I | N | Z | C |
|---|---|---|---|---|---|---|---|
| ↕ | 1 | 1 | ↕ | — | ↕ | ↕ | ↕ |

V: $A7\&M7\&\overline{R7} \mid \overline{A7}\&\overline{M7}\&R7$
Set if a two's complement overflow resulted from the operation; cleared otherwise

H: $A3\&M3 \mid M3\&\overline{R3} \mid \overline{R3}\&A3$
Set if there was a carry from bit 3; cleared otherwise

N: R7
Set if MSB of result is 1; cleared otherwise

Z: $\overline{R7}\&\overline{R6}\&\overline{R5}\&\overline{R4}\&\overline{R3}\&\overline{R2}\&\overline{R1}\&\overline{R0}$
Set if result is $00; cleared otherwise

C: $A7\&M7 \mid M7\&\overline{R7} \mid \overline{R7}\&A7$
Set if there was a carry from the MSB of the result; cleared otherwise

**SUB** **Subtract** **SUB**

**Operation**

A ← (A) – (M)

**Description**

Subtracts the contents of M from A and places the result in A

**Condition Codes and Boolean Formulae**

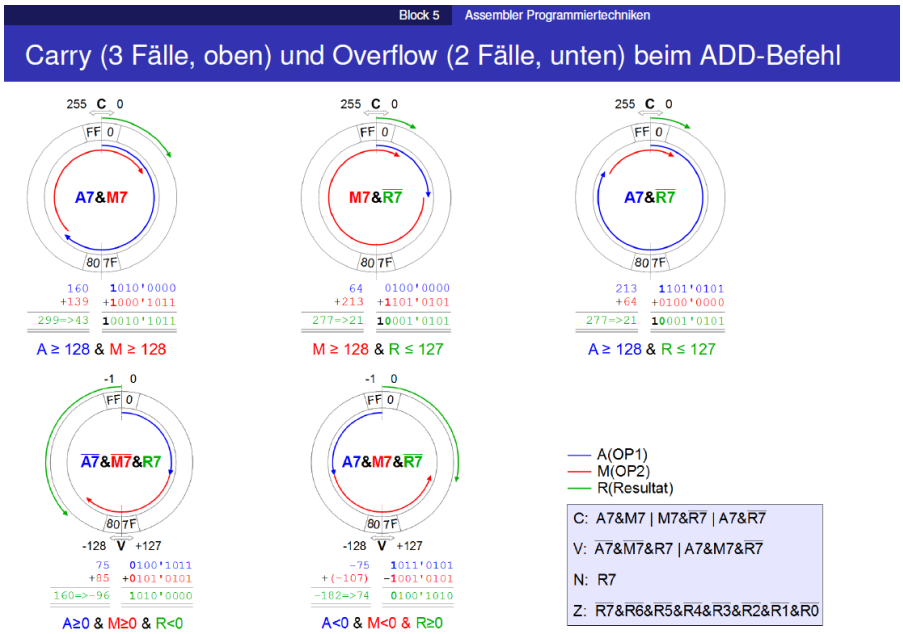
| V | | | H | I | N | Z | C |
|---|---|---|---|---|---|---|---|
| ↕ | 1 | 1 | — | — | ↕ | ↕ | ↕ |

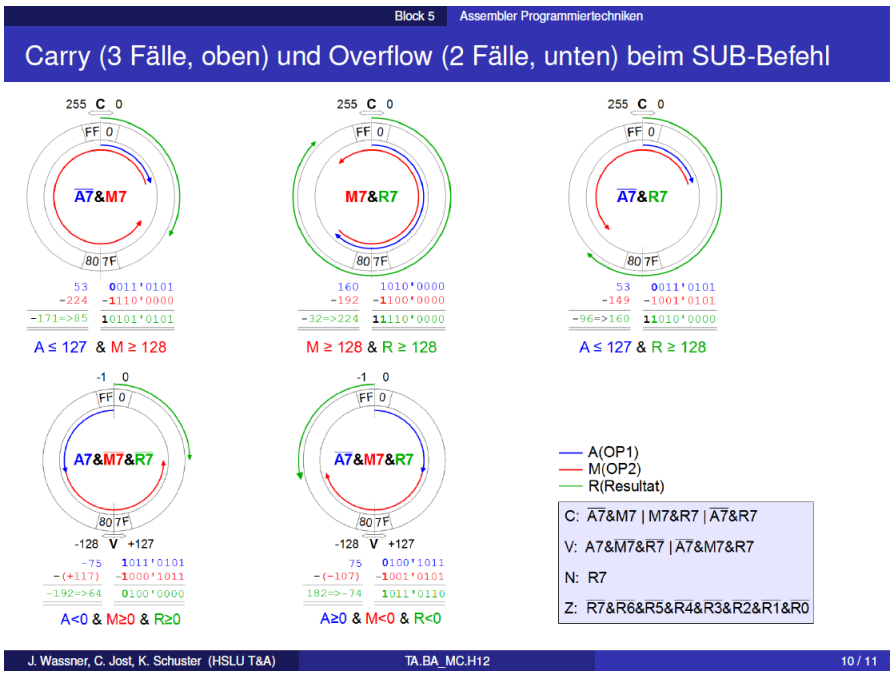
V: $A7\&\overline{M7}\&\overline{R7} \mid \overline{A7}\&M7\&R7$
Set if a two's complement overflow resulted from the operation; cleared otherwise. Literally read, an overflow condition occurs if a positive number is subtracted from a negative number with a positive result, or, if a negative number is subtracted from a positive number with a negative result.

N: R7
Set if MSB of result is 1; cleared otherwise

Z: $\overline{R7}\&\overline{R6}\&\overline{R5}\&\overline{R4}\&\overline{R3}\&\overline{R2}\&\overline{R1}\&\overline{R0}$
Set if result is $00; cleared otherwise

C: $\overline{A7}\&M7 \mid M7\&R7 \mid R7\&\overline{A7}$
Set if the unsigned value of the contents of memory is larger than the unsigned value of the accumulator; cleared otherwise



6. Schreiben Sie einen Programmteil, welcher 2 Operanden von den Adressen $00C0 und $00C1 einliesst und nach einer noch anzugebenden arithmetischen Verknüpfung (Aufgabe 1.3 und 1.4) das Resultat ins X-Register schreibt und die aktuellen Werte der Flags VNZC auf 4 LEDs wie folgt ausgibt:
   C → PTF0 = LED B FL
   Z → PTF1 = LED R FL
   V → PTE7 = LED B FR
   N → PTF2 = LED R FR
   *Die LED sind schön 1:1 wie die Flags im CCR sind. D.h. man muss die Flags / Bits nicht schieben und kann sie direkt auf den Port F resp. E ausgeben.*

```
; Rear-LED ON, damit sichtbar ist, dass der MC-Car noch läuft (Save Battery)
;---------------------------------------------------------------------
          LDA   #$04                    ; LED Rear On --> uP still running ;-)
          STA   PTDDD
          STA   PTDD

; Start des Benutzer-Codes
;---------------------------------------------------------------------

userMain: LDA   #$07
          STA   PTFDD                   ; Pin0-3, Port F DataDirection = Output
          LDA   #$80
          STA   PTEDD                   ; Pin 7, Port E DataDirection

          LDA   #$FF                    ; All LED OFF
          STA   PTED
          STA   PTFD

          JMP   AUFGABE1                ; Jump to Aufgabe X


 ;----- Aufgabe 4.1 ----------------------------------------------------------
AUFGABE1: LDA   #$00                    ; Default-Wert 00 in Speicher C0 und C1 speichern
          STA   $00C0
          STA   $00C1

Loop:     LDA   $00C0                   ; Load OP1
          ADD   $00C1                   ; ADD OP2
          ;SUB   $00C1                   ; SUB OP2
          TAX                           ; Result -> X // Transfer Accumulator to X
          TPA                           ; Transfer Processor Status Byte to Accumulator
                                        ; (CCR -> Akku)
          COMA                          ; Invertiere Akku (CCR) because of low-active LED
          STA   PTED                    ; V -> PTE7
          STA   PTFD                    ; NZC -> PTF2..0
          BRA   Loop                    ; Endlosschleife
```

7. Benutzen Sie den unter 1.2 vorbereiteten Programmteil, um die Flagsetzung beim ADD Befehl zu analysieren. Legen Sie dazu vor dem Programmablauf zweckmässige Operanden auf die Adressen $00C0 und $00C1 (z.B mit dem Debugger Befehl >mem). Verifizieren Sie die erwarteten Flags auf der Anzeige der roten und blauen FrontLEDs und stellen Sie den Bezug zu den Zahlenkreisdiagrammen auf den Folien her.
   *siehe oben!*

8. Machen Sie dasselbe wie in Aufgabe 1.3 für den Befehl SUB.
   *siehe oben!*

# Bitmanipulationen

9. Schreiben Sie ein kleines Programm, welches das Carry-Flag im CCR mittels Bit-Maskierung löscht! Kontrollieren Sie Ihr Programm, indem Sie das Carry-Flag zuerst mit >reg setzen.

```
TPA
AND #$FE
TAP
```

```
; Rear-LED ON, damit sichtbar ist, dass der MC-Car noch läuft (Save Battery)
;---------------------------------------------------------------------
          LDA   #$04                  ; LED Rear On --> uP still running ;-)
          STA   PTDDD
          STA   PTDD

; Start des Benutzer-Codes
;---------------------------------------------------------------------

userMain: LDA   #$07
          STA   PTFDD                 ; Pin0-3, Port F DataDirection = Output
          LDA   #$80
          STA   PTEDD                 ; Pin 7, Port E DataDirection

          LDA   #$FF                  ; All LED OFF
          STA   PTED
          STA   PTFD

          JMP   AUFGABE2              ; Jump to Aufgabe 1 / 2 or 3


;----- Aufgabe 4.2 ----------------------------------------------------------------
          ; Carry-Flag im CCR mittels Bit-Maskierung löschen
          ; vorher Flag setzen, z.B. mittels Debugger Shell: reg SR=0x01
AUFGABE2: NOP                         ; set Carry-Flag now --> SR=0X01
          TPA                         ; CCR -> Akku (Processor Status Byte)
          AND   #$FE                  ; Clear Carry-Flag (Bit0) %00000001
          TAP                         ; Akku -> CCR

          ; Overflow-Flag mittels Bit-Maskierung setzen: reg SR=0x00
          NOP                         ; set Overflow-Flag now --> SR=0x00
          TPA                         ; CCR -> Akku
          ORA   #$80                  ; Set Overflow-Flag (Bit7) %10000000
          TAP                         ; Akku -> CCR

          ; direkte setzen des Carry-Flag
          SEC                         ; Set Carry-Flag

          ; direkte löschen des Carry-Flag
          CLC                         ; Clear Carry-Flag

EndLoop2: BRA   AUFGABE2              ; Endlos-Loop
```

10. Schreiben Sie ein kleines Programm, welches das Overflow-Flag mittels Bit-Maskierung setzt!

*siehe oben!*

```
TPA
ORA #$80
TAP
```

11. Es gibt einige Befehle, welches das direkte Setzen (Sxx von SET) bzw. Löschen (Cxx von CLEAR) zulassen. Wie heissen die Befehle, welche dasselbe wie unter Aufgabe 2.1 und 2.2 tun?

```
SEC   ; Set Carry-Flag
CLC   ; Clear Carry-Flag
```

# BRANCH-Befehle

12. Informieren Sie sich im HSLU_HCS08_Programming_Guide über die BRANCH-Befehle.

```
; BCC       Branch if Carry Bit Clear (C = 0)
; BCS       Branch if Carry Bit Set (C = 1)
; BNE       Branch if Not Equal (Z = 0)
; BEQ       Branch if Equal (Z = 1)
; BHCC      Branch if Half Carry Bit Clear (H = 0)
; BHCS      Branch if Half Carry Bit Set (H = 1)
; BMC       Branch if Interrupt Mask Clear (I = 0)
; BMS       Branch if Interrupt Mask Set (I = 1)
; BPL       Branch if Plus (N = 0)
; BMI       Branch if Minus (N = 1)
; BRCLR n   Branch if Bit n in Memory Clear
; BRSET n   Branch if Bit n in Memory Set

; BGE       Branch if Greater Than or Equal To (N and V = 0)
; BGT       Branch if Greater Than (Z or (N and V) = 0)
; BHI       Branch if Higher (C or Z = 0)
; BHS       Branch if Higher or Same (Same as BCC)
; BLE       Branch if Less Than or Equal To ( Z or (N and V) = 1)
; BLO       Branch if Lower (C = 1)
; BLS       Branch if Lower or Same (C or Z = 1)
; BLT       Branch if Less Than (signed) (N and V = 1)
```

Wie heissen und wie funktionieren die Branch-Befehle mit denen man gleichzeitig maskieren kann?

```
BRCLR  n, Addr, Label       ; Verzweigt nach Label, wenn Bit n der Speicherstelle
                            ; an Adresse Addr nicht gesetzt ist (Addr nur DIR)

BRSET  n, Addr, Label       ; Verzweigt nach Label, wenn Bit n der Speicherstelle
                            ; an Adresse Addr gesetzt ist (Addr nur DIR)
```

13. Schreiben Sie ein kleines Programm, welches mittels eines Compare-Befehls zwei 8-Bit Operanden vergleicht und dann mittels verschiedener BRANCH Befehle verzweigt. Machen Sie sich so die Bedeutung verschiedener Branch Befehle klar (BCS, BHI, etc.).
Signalisation: Kein Sprung Schalte LED B FL ein (PTF0)
Sprung Schalte LED B FR ein (PTE7)

```
;----- Aufgabe 4.3 ----------------------------------------------------------
AUFGABE3: LDA   #$FF                    ; All LED OFF
          STA   PTED
          STA   PTFD

          LDA   #$0F                    ; Load $0F into Akku
          CMP   #$0F                    ; Compare with $0F
          BEQ   ISEQUAL                 ; Branch if Equal -> JMPEQUAL (Z = 0)
NOTEQUAL: ; not Equal -> Schalte LED B FL ein (PTF0)
          LDA   #$FE                    ; LED Low-Active! 01 -> FE
          STA   PTFD
          BRA   AUFGABE3                ; Loop
ISEQUAL:  ; Equal ->  Schalte LED B FR ein (PTE7)
          LDA   #$7F                    ; LED Low-Active! 80 -> 7F
          STA   PTED
          BRA   AUFGABE3                ;Loop
```

# Unterprogramme, Stack und Parameter

## 14. Aufruf einer leeren Funktion, ohne Parameter

```
void function(void)
{
   asm NOP;
}

/**
 * Hauptprogramm
 */
void main(void)
{
   function();
   for(;;){}
}
```

*Unterprogrammaufruf und Parameterübergabe:*

| | | | | nächster Befehl | |
|---|---|---|---|---|---|
| **Trace** | **PC** | **SP** | **HX** | **C** | **Ass** |
| BP: | 1AF8 | 2FF | | function1() { | BSR -2 |
| 1 | 1AF6 | 2FD | | asm NOP | NOP |
| 2 | 1AF7 | 2FD | | } | RTS |
| 3 | 1AFA | 2FF | | for(;;) {} | BRA +0 |

*Stack:*

| | | : | SP nach Trace: |
|---|---|---|---|
| | $02FB | | |
| | $02FC | | |
| | $02FD | | 1, 2 |
| | $02FE | 1A | |
| Bottom | $02FF | FA | BP |
| | | : | |

# 15. Aufruf einer Funktion mit lokalen Variablen, ohne Parameterübergabe

```
void function(void)
{
   unsigned char ucl;
   int il;

   ucl = 1;
   il = (int)ucl;
   ucl++;
}

/**
 * Hauptprogramm
 */
void main(void)
{
   function();
   for(;;){}
}
```

*Unterprogrammaufruf und Parameterübergabe:*

| | | | | nächster Befehl | |
|---|---|---|---|---|---|
| **Trace** | **PC** | **SP** | **HX** | **C** | **Ass** |
| BP: | 1B08 | 2FF | n.a. | function2() { | BSR -18 |
| 1 | 1AF6 | 2FD | | function2(void)<br>unsigned char uc1<br>int 1l | AIS #-3 |
| 2 | 1AF8 | 2FA | | uc1 = 1 | LDA #0x01 |
| 3 | 1AFA | | | | TSX |
| 4 | 1AFB | | 02FB | | STA, X |
| 5 | 1AFC | | | il = (int)uc1 | LDX #0x01 |
| 6 | 1AFE | | 0201 | | CLRH |
| 7 | 1AFF | | 0001 | | STHX 2,SP |
| 8 | 1B02 | | | ucl++ | INCA |
| 9 | 1B03 | | | | TSX |
| 10 | 1B04 | | 2FB | | STA, X |
| 11 | 1B05 | | | } | AIS #3 |
| 12 | 1B07 | 2FD | | | RTS 0 |
| 13 | 1B0A | 2FF | | for(;;) {} | BRA +0 |
| 14 | | | | | |

*Stack:*

| | | : | SP nach Trace: |
|---|---|---|---|
| | $02F9 | uu | |
| | $02FA | uu | 2 |
| ucl | $02FB | ~~01~~ 02 | |
| il-H | $02FC | 00 | |
| il-L | $02FD | 01 | 1, 12 |
| | $02FE | 1B | |
| Bottom | $02FF | 0A | BP, 13 |

## 16. Aufruf einer Funktion mit lokalen Variablen, mit Parametern und Rückgabe

```
char function (unsigned char ucVal, unsigned char *ucRef)
{
   unsigned char ucl1;
   ucl1 = ucVal;
   ucl1++;
   (*ucRef)++; /* Achtung: Ohne Klammern, wird der Pointer inkrementiert */
   return ucl1;
}

/**
 * Hauptprogramm
 */
void main(void)
{
   unsigned char uc1=0, uc2=0, uc3;
   uc3 = function(uc1, &uc2);
   uc3++;
   for(;;){}
}
```

*Unterprogrammaufruf und Parameterübergabe:*

| | | | | nächster Befehl | |
|---|---|---|---|---|---|
| **Trace** | **PC** | **SP** | **HX** | **C** | **Ass** |
| BP: | 1B02 | 02FF | n.a. | main() | AIS #-3 |
| 1 | 1B04 | 02FC | n.a. | uc1=0 | TSX |
| 2 | 1B05 | | 02FD | | CLR 2,X |
| 3 | 1B07 | | | uc2=0, uc3 | CLR, X |
| 4 | 1B08 | | | uc3 = function3(.. | CLR A |
| 5 | 1B09 | | | | BSR -19 |
| 6 | 1AF6 | 02FA | | char function3(... | PSH H |
| 7 | 1AF7 | 02F9 | | ucl1 = ucVal | STA 1, SP |
| 8 | 1AFA | | | ucl1++ | INC 1, SP |
| 9 | 1AFD | | | (*ucRef)++ | INC, X |
| 10 | 1AFE | | | return ucl1(;;) {} | TSX |
| 11 | 1AFF | | 02FA | | LDA, X |
| 12 | 1B00 | | | } | PUL H |
| 13 | 1B01 | 02FA | 01FA | | RTS |
| 14 | 1B0B | 02FC | | uc3 = function3 (... | TSX |
| 15 | 1B0C | | 02FD | | STA 1,X |
| 16 | 1B0E | | | uc3++ | INC 1,X |
| 17 | 1B10 | | | for (;;) {} | BRA +0 |

*Stack:*

| | | : | SP nach Trace: |
|---|---|---|---|
| | $02F9 | uu | 7 |
| ucl1 | $02FA | ~~02~~ ~~00~~ 01 | 6, 13 |
| | $02FB | 1B | |
| | $02FC | 0B | 1, 14 |
| uc2 | $02FD | ~~00~~ 01 | |
| uc3 | $02FE | ~~01~~ 02 | |
| uc1<br>Bottom | $02FF | 00 | BP |

## Kontrollfragen:

17. Wie werden die Parameter ucVal und ucRef beim Funktionsaufruf übergeben?
    *ucVal wird via Akku (Register) übergeben*
    *ucRef wird via Stack übergeben*

18. Wie heissen diese Arten der Parameterübergabe?
    *ucVal: call-by-value*
    *ucRef: call-by-reference*

19. An welchen Adressen liegen die Variablen uc1, uc2, uc3 und ucl1? Warum?
    *uc1: $02FF*
    *uc2: $02FD*
    *uc3: $02FE*
    *ucl1: $02FA*
    *Reihenfolge wie sie deklariert / übergeben wurden.*

20. Wie wird das Resultat der Funktion ans Hauptprogramm übergeben?
    *Via Akku*

21. Wofür werden im Programm die Befehle PSH H und PUL H verwendet?
    *Speicher auf dem Stack reservieren / freigeben*

22. Welcher andere Befehl wäre möglich?
    *PSHH → AIS -1*
    *PULH → AIS +1*
    *(AIS: Add Immediate Value (Signed) to Stack Pointer)*

23. Warum wird hier PSH H und PUL H verwendet?
    *Es wird weniger Programm-Speicher benötigt.*
    *PSHH / PULH sind 1-Byte Befehle, AIS braucht noch einen zweiten Operanden → 2-Byte*

# Timersystem mit und ohne Interrupt

24. 1-Sekunden Timer mit Overflow-Flag (TOF) Polling
Realisieren Sie einen 1-Sekunden Timer (LED am Port F:1 blinken lassen) indem zwischen jedem Toggeln der LED eine Sekunde gewartet wird. Die 1000 ms Verzögerung soll durch Abfrage (Polling) des Timer Overflow Flags (TOF) von TPM1 realisiert werden.

```
/*************************************************************
*** Uebung 6.1
*** 1-Sekunden Timer mit Overflow-Flag (TOF) Polling
*************************************************************/
#include "platform.h" /* include peripheral declarations */

/**
 * Application entry point.
 */
void main(void)
{
  // Realisieren Sie einen 1-Sekunden Timer (LED am Port F:1 blinken lassen)
  // indem zwischen jedem Toggeln der LED eine Sekunde gewartet wird.
  // Die 1000 ms Verzögerung soll durch Abfrage (Polling) des
  // Timer Overflow Flags (TOF) von TPM1 realisiert werden.

  // init Port
  PTFDD_PTFDD1 = 1; // PTD1 als Ausgang
  PTFDD_PTFDD2 = 1; // PTD2 als Ausgang
  PTFD = 2;         // init LED

  // init TPM
  TPM1SC_PS    = 4;     // Prescaler: 16
                        // (16.3.1 TPM Status and Control Register (TPMxSC), S.281)
  TPM1MOD      = 62499; // 1'000'000 : 16 = 62500 - 1
  TPM1SC_TOF   = 0;     // TOF-Flag zurücksetzen
  TPM1SC_CLKSx = 2;     // Bin: 0x10 - Use Fix Syst.Clk - 1MHz (1 micros)
                        // --> Timer starten

  for(;;)
  {
    if (TPM1SC_TOF) {
      TPM1SC_TOF = 0;           // TOF Flag reset
      PTFD_PTFD1 = !PTFD_PTFD1; // LED R FL
      PTFD_PTFD2 = !PTFD_PTFD2; // LED R FR
    }

    __RESET_WATCHDOG(); /* feeds the dog */
  } /* loop forever */
} // main
```

## 25. 1-Sekunden Timer mit Overflow-Flag (TOF) Interrupt

Realisieren Sie einen 1-Sekunden Timer (LED am Port F:1 blinken lassen) indem Sie jede Sekunde einen Timer-Overflow Interrupt von TPM1 auslösen lassen.

### main.c

```
/*************************************************************
*** Uebung 6.2
*** Sekunden Timer mit Overflow-Flag (TOF) Interrupt
*************************************************************/
#include "platform.h" /* include peripheral declarations */

/*
 * Interrupt Funktion: LED blinken
 * Interrupt Funktion in Project.prm definieren !!!
 *   >>> Project Settings > Linker Files > Project.prm
 *   >>> VECTOR ADDRESS 0xFFE0 toggleLED         // TPM1 overflow
 */
interrupt void toggleLED(void)
{ // TOF muss nicht abgefragt werden, sonst hätte es ja kein Interrupt gegeben
  TPM1SC_TOF = 0;           // TOF-Flag reset
  PTFD_PTFD1 = !PTFD_PTFD1; // LED R FL
  PTFD_PTFD2 = !PTFD_PTFD2; // LED R FR
}

/**
 * Application entry point.
 */
void main(void)
{
  // Realisieren Sie einen 1-Sekunden Timer (LED am Port F:1 blinken lassen) indem
  // Sie jede Sekunde einen Timer-Overflow Interrupt von TPM1 auslösen lassen.

  // init Port
  PTFDD_PTFDD1 = 1; // PTD1 als Ausgang
  PTFDD_PTFDD2 = 1; // PTD2 als Ausgang
  PTFD = 2;         // init LED

  // init TPM
  TPM1SC_PS    = 4;     // Prescaler: 16
                        // (16.3.1 TPM Status and Control Register (TPMxSC), S.281)
  TPM1MOD      = 62499; // 1'000'000 : 16 = 62500 - 1
  TPM1SC_TOF   = 0;     // TOF-Flag zurücksetzen
  TPM1SC_TOIE  = 1;     // Enable Interrupt
  TPM1SC_CLKSx = 2;     // Bin: 0x10 - Use Fix Syst.Clk - 1MHz (1 micros)
                        // --> Timer starten

  EnableInterrupts;     // Enable Global Interrupt

  for(;;)
  {
   // do nothing here - our interrupt does the work...

    __RESET_WATCHDOG(); /* feeds the dog */
  } /* loop forever */
} // main
```

### project.prm

[…]

```
VECTOR ADDRESS 0xFFE0 toggleLED         // TPM1 overflow
```

[…]

# Timersystem im Output-Compare Modus

Sie verstehen die Betriebsart Output-Compare des Timersystems im HCS08. Sie können das Timersystem zur Generierung von akustischen Signalen einsetzen.

## 26. Timer mit Output Compare

Legen Sie in CW ein neues C-Projekt mit Copy/Paste an und nutzen Sie das gegebene File main.c als Hauptdatei. Fügen Sie in der Bibliothek MC_Library zu den Verzeichnissen Lib_Headers bzw. Lib_Sources die gegebenen Files sound.h bzw. sound.c zu.

Analysieren Sie den Code und implementieren Sie im File sound_temp.c die nötigen Konfigurationen für das Timer-Modul TPM1 (siehe CW Task-View), so dass das in main.c gegebene Soundfile abgespielt wird.

**Project.prm**

```
STACKSIZE 0x200    // Stacksize 0x200 => 512 Bytes

VECTOR ADDRESS 0xFFC4 errISR_RTC        // RTC
VECTOR ADDRESS 0xFFC6 errISR_IIC        // IIC
VECTOR ADDRESS 0xFFC8 errISR_ACMP       // ACMP
VECTOR ADDRESS 0xFFCA errISR_ADC        // ADC Conversion
VECTOR ADDRESS 0xFFCC errISR_KBI        // KBI Keyboard
VECTOR ADDRESS 0xFFCE errISR_SCI2T      // SCI2 transmit
VECTOR ADDRESS 0xFFD0 errISR_SCI2R      // SCI2 receive
VECTOR ADDRESS 0xFFD2 errISR_SCI2E      // SCI2 error
VECTOR ADDRESS 0xFFD4 errISR_SCI1T      // SCI1 transmit
VECTOR ADDRESS 0xFFD6 errISR_SCI1R      // SCI1 receive
VECTOR ADDRESS 0xFFD8 errISR_SCI1E      // SCI1 error
VECTOR ADDRESS 0xFFDA errISR_TPM2O      // TPM2 overflow
VECTOR ADDRESS 0xFFDC errISR_TPM2CH1    // TPM2 channel 1
VECTOR ADDRESS 0xFFDE errISR_TPM2CH0    // TPM2 channel 0
VECTOR ADDRESS 0xFFE0 errISR_TPM1O      // TPM1 overflow
       //VECTOR ADDRESS 0xFFE2 errISR_TPM1CH5    // TPM1 channel 5
VECTOR ADDRESS 0xFFE2 soundISRfreq      // TPM1 channel 5
VECTOR ADDRESS 0xFFE4 errISR_TPM1CH4    // TPM1 channel 4
VECTOR ADDRESS 0xFFE6 errISR_TPM1CH3    // TPM1 channel 3
       //VECTOR ADDRESS 0xFFE8 errISR_TPM1CH2    // TPM1 channel 2
VECTOR ADDRESS 0xFFE8 soundISRduration  // TPM1 channel 2
VECTOR ADDRESS 0xFFEA errISR_TPM1CH1    // TPM1 channel 1
VECTOR ADDRESS 0xFFEC errISR_TPM1CH0    // TPM1 channel 0
VECTOR ADDRESS 0xFFF0 errISR_USB        // USB
VECTOR ADDRESS 0xFFF2 errISR_SPI2       // SPI2
VECTOR ADDRESS 0xFFF4 errISR_SPI1       // SPI1
VECTOR ADDRESS 0xFFF6 errISR_MCGLOL     // MCGLOL (Loss of Lock)
VECTOR ADDRESS 0xFFF8 errISR_LowVoltage // Low voltage detect
VECTOR ADDRESS 0xFFFA errISR_IRQ        // IRQ
VECTOR ADDRESS 0xFFFC errISR_SWI        // SWI
VECTOR ADDRESS 0xFFFE _Startup          // Reset vector: this is the default entry point
for an application.
```

## main.c

```
#include "platform.h"      /* include peripheral declarations */
#include "sound.h"

/**
 * Ring Tone Melody to be played in RTTTL format
 */
const char soundFile[] =
"Swiss:d=4,o=5,b=85:8f,16p,16f,f,a#,8a#,16p,16a,a,p,8f,16p,16f,f,c6,8c6,16p,16a#,a#,p,d6,
8p,8d6,8c6,8c6,c6,8p,8a#,8a,2g,e,2f,p,p";
const char soundFile2[] =
"FinalCountdown:d=4,o=5,b=125:p,8p,16b,16a,b,e,p,8p,16c6,16b,8c6,8b,a,p,8p,16c6,16b,c6,e,
p,8p,16a,16g,8a,8g,8f#,8a,g.,16f#,16g,a.,16g,16a,8b,8a,8g,8f#,e,c6,2b.,16b,16c6,16b,16a,1
b,p,p";

/**
 * Switch on Rear LEDs on Port D2
 */
void initPorts(void)
{
    PTDDD = 0x04;
    PTDD = 0x04;
}


/**
 * TPM1: Counter running with frequency 1 MHz
 * - No TOF interrupt
 * - Modulo = default
 * - Prescale = 1
 */
void initTimer(void)
{
  TPM1SC = 0x10;
}


/**
 * main program
 */
void main(void)
{
  int i = 0;

  initPorts();             // Port init

  initTimer();             // Timer init

  soundPlay(soundFile2);    // Play ring tone melody

  EnableInterrupts;        // Interrupts enable

  for(;;) {

    if (!soundIsPlaying()) {
      if (i == 0) {
        soundPlay(soundFile);
        i++;
      } else {
        soundPlay(soundFile2);
        i = 0;
      }
    }

  }
}
```

## sound.h

```
#ifndef SOUND_H
#define SOUND_H


bool soundIsPlaying(void);
void soundTooglePlayPause(void);

void soundStart(void);
void soundStop(void);

void soundPlay(const char *soundFile);

#endif /* SOUND_H_ */
```

## sound.c

```
#include "platform.h"
#include "sound.h"

#define FCNT              1000000   // TPM1 counter frequency [Hz]
#define MAX_MELODY_SIZE       800   // max number of notes

typedef struct
{
  uint8 note;     // note frequency (0 = a Okt.4 ... 47 = g# Okt.7, 48 = pause)
  uint8 time;     // note duration (in units of 1/64 notes)
} tNote;


const uint16 noteCounterticks[49]=  // # of counter ticks for half the period duration of
some note
{
    // c            c#              d               d#              e
f               f#             g              g#               a              a#
b/h
    FCNT/(2*261.6),  FCNT/(2*277.2),  FCNT/(2*293.7),  FCNT/(2*311.1),  FCNT/(2*329.6),
FCNT/(2*349.2),  FCNT/(2*370),  FCNT/(2*392),  FCNT/(2*415.3),  FCNT/(2*440),
FCNT/(2*466.2),  FCNT/(2*493.9),  // 4. oktave
    FCNT/(2*523.3),  FCNT/(2*554.4),  FCNT/(2*587.3),  FCNT/(2*622.3),  FCNT/(2*659.3),
FCNT/(2*698.5),  FCNT/(2*740),  FCNT/(2*784),  FCNT/(2*830.6),  FCNT/(2*880),
FCNT/(2*932.3),  FCNT/(2*987.8),  // 5. oktave
    FCNT/(2*1046.5), FCNT/(2*1108.7), FCNT/(2*1174.7), FCNT/(2*1244.5), FCNT/(2*1318.5),
FCNT/(2*1396.9), FCNT/(2*1480), FCNT/(2*1568), FCNT/(2*1661.2), FCNT/(2*1760),
FCNT/(2*1864.7), FCNT/(2*1975.5), // 6. oktave
    FCNT/(2*2093),   FCNT/(2*2217.4), FCNT/(2*2349.3), FCNT/(2*2489),   FCNT/(2*2637),
FCNT/(2*2793.8), FCNT/(2*2960), FCNT/(2*3136), FCNT/(2*3322.4), FCNT/(2*3520),
FCNT/(2*3729.4), FCNT/(2*3951),   // 7. oktave
    65535 //pause
};


static uint16 tick64;    // # of counter ticks for a 1/64 note

static char soundTitle[20];
static tNote melody[MAX_MELODY_SIZE];
static uint16 melodyPos;
static uint16 melodySize;
static uint16 delay;
```

```
/**
 * ISR: TPM1 Channel 2 - Output Compare
 * Realizes tone duration by counting down the
 * number of 1/64 notes of current tone
 */
interrupt void soundISRduration(void)
{
  // @TODO complete ISR TPM1 Ch2
  TPM1C2SC_CH2F = 0;
  TPM1C2V += tick64; // Counter: dauer für 1/64

  if (delay > 0)  // current tone not complete yet
  {
    delay--;
  }
  else            // current tone complete, setup to play next tone
  {
    melodyPos++;
    if (melodyPos < melodySize)
    {
      TPM1C5V = TPM1CNT + noteCounterticks[melody[melodyPos].note]; // next frequency
      delay = melody[melodyPos].time * 4;                           // next duration

      // tone: toggle output, pause (=48): do not toggle output
      TPM1C5SC_ELS5A = (melody[melodyPos].note == 48 ? 0 : 1);
    }
    else
    {
      // end of melody : do not toggle output anymore
      TPM1C5SC = 0;
      TPM1C2SC = 0;
    }
  }
}


/**
 * ISR: TPM1 Channel 5 - Output Compare
 * Realizes tone frequency by toggling output pin
 * every half period duration of current tone
 */
interrupt void soundISRfreq(void)
{
  // @TODO write ISR TPM1 Ch5
  TPM1C5SC_CH5F = 0; // interrupt flag reset
  // CV = current CV + frequency counterTick
  TPM1C5V += noteCounterticks[melody[melodyPos].note]; // CounterTicket curr. frequency
}
```

```
/**
 * Starts a melody from the current position
 *
 * @remarks: Initializes TPM1 as follows:
 *   CH2 : Output Compare without port pin use, Interrupt enable
 *   CH5 : Output Compare, toggle port pin, Interrupt enable
 */
void soundStart(void)
{
  if (melodyPos == melodySize) melodyPos = 0;

  // @TODO Write TPM1 init function
  // configure "freq" timer channel
  // TPM1 channel 5
  TPM1C5V = TPM1CNT + noteCounterticks[melody[melodyPos].note]; // CounterTicket
  TPM1C5SC_MS5x = 1;   // Channel Mode: Output Compare
  TPM1C5SC_ELS5x = 1;  // toggle port pin
  TPM1C5SC_CH5F = 0;   // clear interrupt flag
  TPM1C5SC_CH5IE = 1;  // Enable Interrupt
  //TPM1C5SC = 0x54;

  // configure "delay" timer channel
  // TPM1 channel 2
  delay = melody[melodyPos].time; // die anzahl 64tel in Delay speichern
  TPM1C2V = TPM1CNT + tick64; // Counter: dauer für 1/64
  TPM1C2SC_MS2x = 1;   // Channel Mode: Output Compare
  TPM1C2SC_ELS2x = 0;  // without port pin use (software compare)
  TPM1C2SC_CH2F = 0;   // TOF-Flag reset
  TPM1C2SC_CH2IE = 1;  // Enable Interrupt
}


/**
 * Stops the current melody
 */
void soundStop(void)
{
  TPM1C5SC = 0;
  TPM1C2SC = 0;
}


/**
 * Returns true, if the player is playing a meldoy
 *
 * @returns
 *    true, if playing, false else
 */
bool soundIsPlaying(void)
{
  return (TPM1C5SC != 0);
}


/**
 * Toggles between play and pause.
 */
void soundTooglePlayPause(void)
{
  if (soundIsPlaying()) soundStop();
  else soundStart();
}
```

```
/*
 * Parses the ring tone string and stores frequency and duration
 * for every tone in an array of type tNote.
 *
 *  @param [in] *soundFile   - RTTTL string
 */
void soundPlay(const char *p)
{
  uint8 i, value, param, note;
  uint8 duration, defDuration;
  uint8 octave, defOctave;

  // Read Title
  for (i=0; i<sizeof(soundTitle); i++)
  {
    if (*p == ':') break;
    soundTitle[i] = *p;
    p++;
  }

  while (*p && *p != ':') p++;
  if (!*p) return;
  p++;


  // parse default values
  while (*p) {
    while (*p == ' ') p++;     // Skip Spaces
    if (!*p) return;           // abort at the end of the string
    if (*p == ':') break;

    param = *p++;
    if (*p != '=') return;

    p++;
    value = 0;
    while (*p >= '0' && *p <= '9') value = value * 10 + (*p++ - '0');

    switch (param) {
      case 'd': defDuration = 64 / value; break;
      case 'o': defOctave = ((uint8)value - 4) * 12; break;
      case 'b': tick64 = (uint16)(((60 * 10000000l) / (value * 64)) - 1); break; // bpm
    }

    while (*p == ' ') p++;
    if (*p == ',') p++;
  }
  p++;

  melodySize = 0;
  while (*p)
  {
    duration = defDuration;
    octave = defOctave;

    // Skip whitespace
    while (*p == ' ') p++;
    if (!*p) return;

    // Parse duration
    if (*p >= '0' && *p <= '9')
    {
     value = 0;
     while (*p >= '0' && *p <= '9') value = value * 10 + (*p++ - '0');
     duration = (uint8) (64 / value);
    }
```

```
// Parse note
    switch (*p){
      case 0: return;
      case 'C': case 'c': note = 0; break;
      case 'D': case 'd': note = 2; break;
      case 'E': case 'e': note = 4; break;
      case 'F': case 'f': note = 5; break;
      case 'G': case 'g': note = 7; break;
      case 'A': case 'a': note = 9; break;
      case 'H': case 'h': note = 11; break;
      case 'B': case 'b': note = 11; break;
      case 'P': case 'p': note = 48; break;
    }
    p++;

    if (*p == '#'){
       note++;
       p++;
    }

    if (*p == 'b'){
       note--;
       p++;
    }

    // Parse special duration
    if (*p == '.'){
      duration += duration / 2;
      p++;
    }

    // Parse octave 4..7
    if (*p >= '0' && *p <= '9'){
      octave = ((*p++ - '0') - 4) * 12;
    }

    // Parse special duration (again...)
    if (*p == '.') {
      duration += duration / 2;
      p++;
    }

    // Skip delimiter
    while (*p == ' ') p++;
    if (*p == ',') p++;

    note += octave;
    if (note > 48) note = 48;

    melody[melodySize].note = note;
    melody[melodySize].time = duration;
    melodySize++;
    if (melodySize >= MAX_MELODY_SIZE) break;
  }

  melodyPos = 0;
  soundStart();
}
```

# 7.2: Timersystem im Input-Capture Modus

Sie verstehen die Betriebsart Input-Capture des Timersystems im HCS08. Sie können das Timersystem zur Auswertung von empfangenen Signalen einsetzen.

## 27. Timer mit Input Capture

Legen Sie in CW ein neues C-Projekt mit Copy/Paste an und nutzen Sie das gegebene File main.c als Hauptdatei. Fügen Sie in der Bibliothek MC_Library zu den Verzeichnissen Lib_Headers bzw. Lib_Sources die gegebenen Files ifrRx.h bzw. ifrRxFront_temp.c zu (Drag/Drop/Copy).

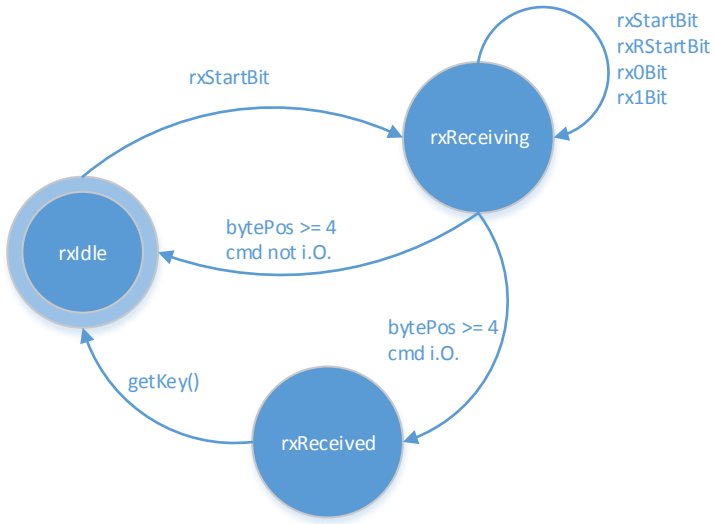
Analysieren Sie den gegebenen Code. Implementieren Sie im File ifrRxFront_temp.c an den 3 gekennzeichneten Stellen (siehe CW Task-View) die fehlende Funktionalität, so dass man mit den Tasten S, W und T der Fernbedienung die Soundwiedergabe wie folgt steuern kann:

S - Start / Pause

W - Skip to next song

T - Skip to previous song

*Statemachine:*




**Project.prm**

```
STACKSIZE 0x200    // Stacksize 0x200 => 512 Bytes

VECTOR ADDRESS 0xFFC4 errISR_RTC        // RTC
VECTOR ADDRESS 0xFFC6 errISR_IIC        // IIC
VECTOR ADDRESS 0xFFC8 errISR_ACMP       // ACMP
VECTOR ADDRESS 0xFFCA errISR_ADC        // ADC Conversion
VECTOR ADDRESS 0xFFCC errISR_KBI        // KBI Keyboard
VECTOR ADDRESS 0xFFCE errISR_SCI2T      // SCI2 transmit
VECTOR ADDRESS 0xFFD0 errISR_SCI2R      // SCI2 receive
VECTOR ADDRESS 0xFFD2 errISR_SCI2E      // SCI2 error
VECTOR ADDRESS 0xFFD4 errISR_SCI1T      // SCI1 transmit
VECTOR ADDRESS 0xFFD6 errISR_SCI1R      // SCI1 receive
VECTOR ADDRESS 0xFFD8 errISR_SCI1E      // SCI1 error
VECTOR ADDRESS 0xFFDA errISR_TPM2O      // TPM2 overflow
VECTOR ADDRESS 0xFFDC errISR_TPM2CH1    // TPM2 channel 1
VECTOR ADDRESS 0xFFDE errISR_TPM2CH0    // TPM2 channel 0
VECTOR ADDRESS 0xFFE0 errISR_TPM1O      // TPM1 overflow
//VECTOR ADDRESS 0xFFE2 errISR_TPM1CH5    // TPM1 channel 5
VECTOR ADDRESS 0xFFE2 soundISRfreq      // TPM1 channel 5
VECTOR ADDRESS 0xFFE4 errISR_TPM1CH4    // TPM1 channel 4
VECTOR ADDRESS 0xFFE6 errISR_TPM1CH3    // TPM1 channel 3
//VECTOR ADDRESS 0xFFE8 errISR_TPM1CH2    // TPM1 channel 2
VECTOR ADDRESS 0xFFE8 soundISRduration  // TPM1 channel 2
VECTOR ADDRESS 0xFFEA errISR_TPM1CH1    // TPM1 channel 1
//VECTOR ADDRESS 0xFFEC errISR_TPM1CH0    // TPM1 channel 0
VECTOR ADDRESS 0xFFEC ifrFrontISR       // TPM1 channel 0
VECTOR ADDRESS 0xFFF0 errISR_USB        // USB
VECTOR ADDRESS 0xFFF2 errISR_SPI2       // SPI2
VECTOR ADDRESS 0xFFF4 errISR_SPI1       // SPI1
VECTOR ADDRESS 0xFFF6 errISR_MCGLOL     // MCGLOL (Loss of Lock)
VECTOR ADDRESS 0xFFF8 errISR_LowVoltage // Low voltage detect
VECTOR ADDRESS 0xFFFA errISR_IRQ        // IRQ
VECTOR ADDRESS 0xFFFC errISR_SWI        // SWI
VECTOR ADDRESS 0xFFFE _Startup          // Reset vector: this is the default entry point for an application.
```

**main.c**

```
#include "platform.h" /* include peripheral declarations */
#include "ifrRx.h"
#include "sound.h"

const char sound1[] =
    "lastxmas:d=4,o=5,b=112:16d6,e6,8p,e6,8d6,8p,8a,8e6,8e6,8f#6,d.6,8b,8b," \
    "8e6,8e6,f#6,d.6,8b,8c#6,8d6,8c#6,2b.,16e6,f#6,8p,e.6,8p,8b,8f#6,8g6,8f#6," \
    "2e6,8d6,8c#6,8d6,8c#6,c#6,8d6,8p,8c#6,8p,2a,16d6,e6,8p,e6,8d6,8p,8a," \
    "8e6,8e6,8f#6,d.6,8b,8b,8e6,8e6,f#6,d.6,8b,8c#6,8d6,8c#6,2b.,16e6,f#6,8p," \
    "e.6,8p,8b,8f#6,8g6,8f#6,2e6,8d6,8c#6,8d6,8c#6,c#6,8d6,8p,8c#6,8p,a";

const char sound2[] =
    "Swiss:d=4,o=5,b=85:8f,16p,16f,f,a#,8a#,16p,16a,a,p,8f,16p,16f,f,c6,8c6,16p," \
    "16a#,a#,p,d6,8p,8d6,8c6,8c6,c6,8p,8a#,8a,2g,e,2f";

const char sound4[] = "Chariots of Fire:d=16,o=5,b=85:8c#,f#.,g#.,a#.,4g#,4f,8p,8c#," \
    "f#.,g#.,a#.,2g#,8p,8c#,f#.,g#.,a#.,4g#,4f,8p,8f,f#.,f.,c#.,2c#";

const char sound3[] = "Toccata:d=16,o=6,b=63:32a7,[.....],1d5";

/**
 * Configures the port F and G as follows:
 * - Port F as output
 * - Port G as input with pull-up enabled
 */
void initPorts(void)
{
    PTFDD = 0x07;           // Port F = Output
    PTFD = 0x07;

    PTGDD = 0x00;           // Port G = Input
    PTGPE = 0xff;           // Port G = Pull-Up enable
}

/**
 * TPM1: Counter running with frequency 1 MHz
 * - No TOF interrupt
 * - Modulo = default
 * - Prescale = 1
 */
void initTimer(void)
{
  TPM1SC = 0x10;
}

/**
 * main program
 */
void main(void)
{
  const char * soundFiles[] = {sound1, sound2, sound3, sound4};
  uint8 count = sizeof(soundFiles) / sizeof(soundFiles[0]);

  uint8 playIndex = 0;

  initPorts();            // Ports konfigurieren

  ifrRxFrontInit();       // init IR receiver

  initTimer();            // Timer init

  EnableInterrupts;       // Interrupts aktivieren

  for(;;)
  {
    switch (ifrRxFrontGetKey())
    {
      case 'S' : // play/pause
        soundTooglePlayPause();
        break;

      case 'W' : // play next melody
        playIndex++;
        if (playIndex >= count) playIndex = 0;
        soundPlay(soundFiles[playIndex]);
        break;

      case 'T' : // play previous melody
        if (playIndex == 0) playIndex = count;
        playIndex--;
        soundPlay(soundFiles[playIndex]);
        break;

      case '+' :  break;
      case '-' :  break;
    }
  }
}
```

## ifrRx.h

```
#ifndef IFRRX_H_
#define IFRRX_H_

#include "platform.h"

#define TIMER_FREQ                1000000.0

#define START_BIT_PULSE_TIME         9000.0e-6           // 9000 us pulse
#define START_BIT_PAUSE_TIME         4500.0e-6           // 4500 us pause
#define RSTART_BIT_PAUSE_TIME        2250.0e-6           // 2250 us pause
#define PULSE_TIME                    560.0e-6           //  560 us pulse
#define PAUSE_1_TIME                 1690.0e-6           // 1690 us pause
#define PAUSE_0_TIME                  560.0e-6           //  560 us pause

#define MIN_TOLERANCE                   0.7
#define MAX_TOLERANCE                   1.3

#define START_BIT_PULSE_LEN_MIN     ((uint16)(TIMER_FREQ * START_BIT_PULSE_TIME * MIN_TOLERANCE + 0.5)-1)
#define START_BIT_PULSE_LEN_MAX     ((uint16)(TIMER_FREQ * START_BIT_PULSE_TIME * MAX_TOLERANCE + 0.5)-1)
#define START_BIT_PAUSE_LEN_MIN     ((uint16)(TIMER_FREQ * START_BIT_PAUSE_TIME * MIN_TOLERANCE + 0.5)-1)
#define START_BIT_PAUSE_LEN_MAX     ((uint16)(TIMER_FREQ * START_BIT_PAUSE_TIME * MAX_TOLERANCE + 0.5)-1)
#define RSTART_BIT_PAUSE_LEN_MIN    ((uint16)(TIMER_FREQ * RSTART_BIT_PAUSE_TIME * MIN_TOLERANCE + 0.5)-1)
#define RSTART_BIT_PAUSE_LEN_MAX    ((uint16)(TIMER_FREQ * RSTART_BIT_PAUSE_TIME * MAX_TOLERANCE + 0.5)-1)
#define PULSE_LEN_MIN               ((uint16)(TIMER_FREQ * PULSE_TIME * MIN_TOLERANCE + 0.5)-1)
#define PULSE_LEN_MAX               ((uint16)(TIMER_FREQ * PULSE_TIME * MAX_TOLERANCE + 0.5)-1)
#define PAUSE_1_LEN_MIN             ((uint16)(TIMER_FREQ * PAUSE_1_TIME * MIN_TOLERANCE + 0.5)-1)
#define PAUSE_1_LEN_MAX             ((uint16)(TIMER_FREQ * PAUSE_1_TIME * MAX_TOLERANCE + 0.5)-1)
#define PAUSE_0_LEN_MIN             ((uint16)(TIMER_FREQ * PAUSE_0_TIME * MIN_TOLERANCE + 0.5)-1)
#define PAUSE_0_LEN_MAX             ((uint16)(TIMER_FREQ * PAUSE_0_TIME * MAX_TOLERANCE + 0.5)-1)

char ifrRxFrontGetKey(void);
void ifrRxFrontInit(void);

#endif /* IFRRX_H_ */
```

## ifrRxFront.c

```
#include "platform.h"
#include "ifrRx.h"

/**
 * Sets a bit of the @ref rxBuf to '1'
 *
 * @param[in] buf
 *      the desired buffer to set the bit
 * @param[in] bytePos
 *      the desired byte [0..(rxBufSize - 1)]
 * @param[in] bitPos
 *      the desired bit [0...7]
 */
#define SetRxBit(buf, bytePos, bitPos)      (buf[bytePos] |= (1 << bitPos))

/**
 * Sets a bit of the @ref rxBuf to '0'
 *
 * @param[in] buf
 *      the desired buffer to clear the bit
 * @param[in] bytePos
 *      the desired byte [0..(rxBufSize - 1)]
 * @param[in] bitPos
 *      the desired bit [0...7]
 */
#define ClearRxBit(buf, bytePos, bitPos)    (buf[bytePos] &= ~(1 << bitPos))

typedef enum
{
  rxIdle,
  rxReceiving,
  rxReceived
} rxStates;

typedef enum
{
  rxStartBit,
  rxRStartBit,
  rx1Bit,
  rx0Bit
} rxBits;

typedef union
{
  uint8 buf[4];
  struct
  {
    uint16 adr;
    uint8 cmd;
    uint8 cmdN;
  } cmd;
} tCommand;
```

```
#define rxBuf   (rxCommand.buf)
static tCommand rxCommand;
static rxStates rxState;


/**
 * Front infrared receiver interrupt service routine
 */
interrupt void ifrFrontISR(void) // TPM1CH0
{
  static uint16 oldTicks = 0;
  static uint16 pulse, pause;
  static uint8 bitPos = 0;
  static uint8 bytePos = 0;
  rxBits rxBit;

  PTFD_PTFD1 = 0;               // switch on led for debug purposes
  TPM1C0SC_CH0F = 0;            // clear interrupt flag

  if (PTED_PTED2) {             // Port E Channel 2 == 1
    // rising edge -> calculate pulse time
    pulse = TPM1C0V - oldTicks; // calculate pulse time
    oldTicks = TPM1C0V;         // set start ticks
    PTFD_PTFD1 = 1;             // switch off led for debug purposes
    return;                     // exit ISR (we need pulse and pause time)
  } else {
    // falling edge -> calculate pause time
    pause = TPM1C0V - oldTicks; // calculate pause time
    oldTicks = TPM1C0V;         // set start ticks
  }

  // state machine
  // Check if Pulse is Start / Repeat Start or Data (0/1) Bit
  if (pulse > PULSE_LEN_MIN && pulse < PULSE_LEN_MAX)
  {
    // Get Value of the Data Bit: 0 / 1
    if (pause > PAUSE_0_LEN_MIN && pause < PAUSE_0_LEN_MAX) rxBit = rx0Bit;
    else if (pause > PAUSE_1_LEN_MIN && pause < PAUSE_1_LEN_MAX)rxBit = rx1Bit;
  }
  else if (pulse > START_BIT_PULSE_LEN_MIN && pulse < START_BIT_PULSE_LEN_MAX)
  {
    // Get Value of Start / Repeat Start
    if (pause > START_BIT_PAUSE_LEN_MIN && pause < START_BIT_PAUSE_LEN_MAX) rxBit = rxStartBit;
    else if (pause > RSTART_BIT_PAUSE_LEN_MIN && pause < RSTART_BIT_PAUSE_LEN_MAX) rxBit = rxRStartBit;
  }

  // Evaluate current state
  switch (rxState)
  {
    // State: Idle
    case rxIdle:
      // check for start bit
      if (rxBit == rxStartBit)
      {
        bitPos = bytePos = 0;   // reset bit/byte position
        rxState = rxReceiving;  // set state to receiving
      }
      else if (rxBit == rxRStartBit) { /* Repeated Start */ }
      break;

    // State: Receiving
    case rxReceiving:
      // check for start / repeated start bit
      if (rxBit == rxStartBit || rxBit == rxRStartBit)
      {
        bitPos = bytePos = 0;   // reset bit/byte position
      }
      else
      {
        // get Bit -> command
        if (rxBit == rx1Bit) SetRxBit(rxBuf, bytePos, bitPos); // set current Bit = 1
        else ClearRxBit(rxBuf, bytePos, bitPos);               // set current Bit = 0

        // check if command read completed
        if (++bitPos > 7)
        {
          bitPos = 0;           // set current Pos = 0
          bytePos++;            // increment byte Pos
          // check if all bytes read successfully
          if (bytePos >= 4)
          {
            // check if the command negated the same as the received negated command -> set State Received
            // verification of the received message
            if ((rxCommand.cmd.cmd ^ rxCommand.cmd.cmdN) == 0xff) rxState = rxReceived;
            else rxState = rxIdle;  // Received failed -> set State Idle
          }
        }
      }
      break;

    // State: Received
    case rxReceived: break;
```

```
    // Default state
    default: rxState = rxIdle; break; // set Idle state
  }

  PTFD_PTFD1 = 1;               // switch off led for debug purposes
}

/**
 * Returns a key or 0 if no key was pressed.
 */
uint8 ifrRxFrontGetKey(void)
{
  volatile uint16 adr; // not used
  uint8 cmd = 0;

  //check if command received
  if (rxState == rxReceived)
  {
    adr = rxCommand.cmd.adr;      // adr is atm not used

    // translate cmd into readable char-cmd (Key-Desc)
    switch (rxCommand.cmd.cmd)
    {
      case 1 : cmd = 'S'; break;  // 1 --> "S-Key"
      case 2 : cmd = 'W'; break;  // 2 --> "W-Key"
      case 3 : cmd = 'T'; break;  // 3 --> "T-Key"
      case 5 : cmd = '+'; break;  // 5 --> "Plus-Key"
      case 4 : cmd = '-'; break;  // 4 --> "Minus-Key"
    }
    rxState = rxIdle;             // set State back to idle
  }
  return cmd;
}

/**
 * This function configures the TPM1 channel 0 as follows:
 * - input capture on both edges
 * - with interrupt enabled
 */
void ifrRxFrontInit(void)
{
  TPM1C0SC_MS0x  = 0;           // Input capture
  TPM1C0SC_ELS0x = 3;           // Capture on rising or falling edge
  TPM1C0SC_CH0F  = 0;           // clear interrupt flag
  TPM1C0SC_CH0IE = 1;           // Enable Interrupt
}
```

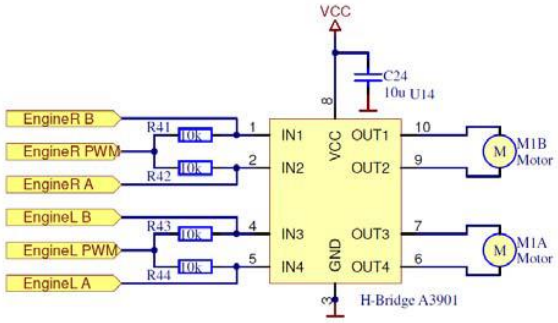
# 7.3: Timersystem im Edge-Aligned PWM Modus

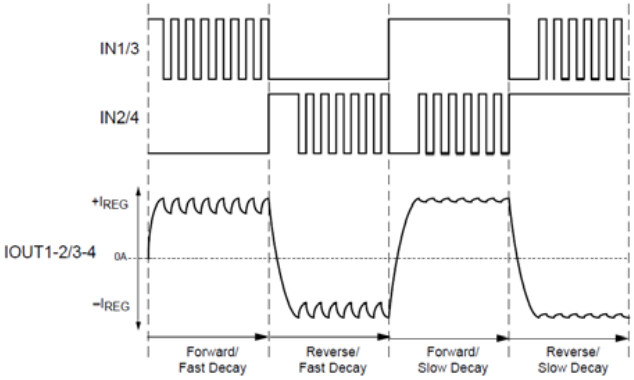
Sie verstehen die Betriebsart Edge-Aligned PWM des Timersystems im HCS08. Sie können das Timersystem zur Ansteuerung von Motoren einsetzen

### 1. Ansteuerung der DC-Motoren

Die beiden DC-Motoren des MC-Car werden über den H-Brücken Treiber A3901 angesteuert. Abbildung 1 zeigt den entsprechenden Ausschnitt aus dem Schema des MC-Car. Der A3901 erfordert die Generierung zweier PWM-Signale an seinen Eingängen IN1/3 für die Vorwärtsbewegung bzw. zweier PWM-Signale an seinen Eingängen IN2/4 für die Rückwärtsbewegung, siehe Abbildung 1 rechts. Da der eingesetzte MC9S08JM60 nicht genügend Timer-Kanäle besitzt, wurde eine Lösung mit insgesamt nur 2 PWM-Signalen und den 4 zusätzlichen Port-Pin Signalen EngineR/L_A/B implementiert.

Beispiel: Es soll der Modus Slow Decay des A3901 genutzt werden. Um in diesem Modus den rechten Motor (M1B) vorwärts drehen zu lassen, konfiguriert man EngineR_A als Input (Signal wird hochohmig) und treibt EngineR_B = 1 als Output. Die Motorgeschwindigkeit wird dann dem Duty Cycle des low-true PWMSignals EngineR_PWM entsprechen.




### 2. Timer mit Output Compare

Legen Sie in CW ein neues C-Projekt mit Copy/Paste an und nutzen Sie das gegebene File main_temp.c als Hauptdatei. Fügen Sie in der Bibliothek MC_Library zu den Verzeichnissen Lib_Headers bzw. Lib_Sources die gegebenen Files motor.h bzw. motor_temp.c zu.

Analysieren Sie den Code und implementieren Sie im File motor_temp.c die nötigen Konfigurationen für das Timer-Modul TPM2 (siehe CW Task-View), so dass die Ansteuerung der Motoren wie oben beschrieben funktioniert. Vervollständigen Sie das File main_temp.c, so dass man über die Tasten der Fernbedienung die Motordrehzahl in kleinen (+/−) bzw. grösseren Schritten (W/T) erhöhen und verringern kann.

**main.c**

```
#include "platform.h"   /* include peripheral declarations */
#include "ifrRx.h"
#include "motor.h"


/**
 * Switch on Rear LEDs on Port D2
 */
void initPorts(void)
{
    PTDDD = 0x04;
    PTDD = 0x04;
}


/**
 * TPM1: Counter running with frequency 1 MHz
 * - No TOF interrupt
 * - Modulo = default
 * - Prescale = 1
 */
void initTimer(void)
{
  TPM1SC = 0x10;
}
```

```
/**
 * main program
 */
void main(void)
{
  initPorts();             // Port init
  initTimer();             // Timer init
  ifrRxFrontInit();        // Infrared init
  motorInit();             // Motor init
  EnableInterrupts;        // Interrupts enable

  for(;;)
  {
    switch (ifrRxFrontGetKey())
    {
      case 'S' :
        motorSetPWMLeft(0);
        motorSetPWMRight(0);
        break;

      case 'W' :
        motorIncrementPWMLeft(20);
        motorIncrementPWMRight(20);
        break;

      case 'T' :
        motorIncrementPWMLeft(-20);
        motorIncrementPWMRight(-20);
        break;

      case '+' :
        motorIncrementPWMLeft(5);
        motorIncrementPWMRight(5);
        break;

      case '-' :
        motorIncrementPWMLeft(-5);
        motorIncrementPWMRight(-5);
        break;
    }
  }
}
```

## motor.h

```
#ifndef MOTOR_H
#define MOTOR_H

int8 motorGetPWMLeft(void);
void motorSetPWMLeft(int8 value);
void motorIncrementPWMLeft(int8 value);

int8 motorGetPWMRight(void);
void motorSetPWMRight(int8 value);
void motorIncrementPWMRight(int8 value);

void motorInit(void);

#endif /* MOTOR_H_ */
```

## motor.c

```
#include <stdlib.h>
#include <string.h>
#include "platform.h"
#include "motor.h"

#define MODULO          127 - 1 // Value 0..127 --> Module must be lower !!!

#define EngineR_A       PTDD_PTDD4
#define EngineR_B       PTDD_PTDD5
#define EngineL_A       PTDD_PTDD6
#define EngineL_B       PTDD_PTDD7

#define EngineR_A_Dir   PTDDD_PTDDD4
#define EngineR_B_Dir   PTDDD_PTDDD5
#define EngineL_A_Dir   PTDDD_PTDDD6
#define EngineL_B_Dir   PTDDD_PTDDD7

static int8 pwmLeft;
static int8 pwmRight;


/**
 * returns the pwm value of the left motor
 *
 * @returns
 *    the value 0..+/-127
 */
int8 motorGetPWMLeft(void)
{
  return pwmLeft;
}

/**
 * returns the pwm value of the right motor
 *
 * @returns
 *    the value 0..+/-127
 */
int8 motorGetPWMRight(void)
{
  return pwmRight;
}

/**
 * increments the speed of the left motor
 *
 * @param [in] value
 *    the desired offset
 */
void motorIncrementPWMLeft(int8 value)
{
  int16 v = pwmLeft + value;
  if (v > 127) v = 127;
  if (v < -127) v = -127;
  motorSetPWMLeft((int8)v);
}

/**
 * increments the speed of the right motor
 *
 * @param [in] value
 *    the desired offset
 */
void motorIncrementPWMRight(int8 value)
{
  int16 v = pwmRight + value;
  if (v > 127) v = 127;
  if (v < -127) v = -127;
  motorSetPWMRight((int8)v);
}

/**
 * sets the speed of the left motor
 *
 * @param [in] value
 *  +1..+127 => speed in forward direction
 *  -1..-127 => speed in backward direction
 *  0 => stop
 */
void motorSetPWMLeft(int8 value)
{
  if(value < 0)               // backward
  {
    EngineL_A_Dir = 0;        // EngineL B = Input = PWM
    EngineL_B_Dir = 1;        // EngineL A = Output
    EngineL_B = 1;

    TPM2C1V = abs(value);

  }
```

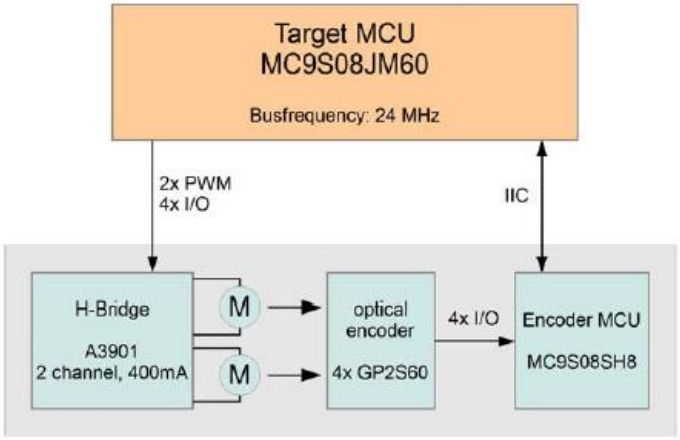
```
  else if(value > 0)          // forward
  {
    EngineL_A_Dir = 1;        // EngineL B = Output
    EngineL_B_Dir = 0;        // EngineL A = Input = PWM
    EngineL_A = 1;

    TPM2C1V = value;
  }
  else                        // stop
  {
    EngineL_A_Dir = 1;        // EngineR A = Output
    EngineL_B_Dir = 1;        // EngineL B = Output
    EngineL_A = 1;
    EngineL_B = 1;

    TPM2C1V = MODULO;
  }
  pwmLeft = value;
}

/**
 * sets the speed of the right motor
 *
 * @param [in] value
 *  +1..+127 => speed in forward direction
 *  -1..-127 => speed in backward direction
 *  0 => stop
 */
void motorSetPWMRight(int8 value)
{
  if(value < 0)               // rückwärts
  {
    EngineR_A_Dir = 1;        // EngineR A = Output
    EngineR_B_Dir = 0;        // EngineR B = Input = PWM
    EngineR_A = 1;

    TPM2C0V = abs(value);
  }
  else if(value > 0)          // vorwärts
  {
    EngineR_A_Dir = 0;        // EngineR A = Input = PWM
    EngineR_B_Dir = 1;        // EngineR B = Output
    EngineR_B = 1;

    TPM2C0V = value;
  }
  else                        // stop
  {
    EngineR_A_Dir = 1;        // EngineR A = Output
    EngineR_B_Dir = 1;        // EngineR B = Output
    EngineR_A = 1;
    EngineR_B = 1;

    TPM2C0V = MODULO;
  }
  pwmRight = value;
}

/**
 * Initializes the motor driver as follows:
 * - TPM2: Clocksource = 24 MHZ, Prescale = 4
 * - TPM2CH0: edge aligned pwm with low-true pulses
 * - TPM2Ch1: edge aligned pwm with low-true pulses
 */
void motorInit(void)
{
  EngineL_A_Dir = 1;
  EngineL_B_Dir = 1;
  EngineR_A_Dir = 1;
  EngineR_B_Dir = 1;

  EngineR_A = 1;
  EngineR_B = 1;
  EngineL_A = 1;
  EngineL_B = 1;

  // Clocksource = 24 MHZ, Prescale = 4
  TPM2SC_CLKSx   = 1;         // Clocksource = 24 MHZ
  TPM2SC_PS      = 2;         // Prescale = 4
  TPM2MOD        = MODULO;    // Module = 126 (MaxValue - 1)

  // Motor Rechts
  // TPM2CH0: edge aligned pwm with low-true pulses
  TPM2C0SC_MS0B  = 1;         // Edge-aligned PWM
  TPM2C0SC_ELS0A = 1;         // Low-true pulses (set output on compare)

  // Motor Links
  // TPM2Ch1: edge aligned pwm with low-true pulses
  TPM2C1SC_MS1B  = 1;         // Edge-aligned PWM
  TPM2C1SC_ELS1A = 1;         // Low-true pulses (set output on compare)
}
```

# 8: IIC-Bussystem / PID

Sie verstehen das IIC-Busprotokoll und können das IIC-Controller Modul des MC9S08JM60 zur Kommunikation mit anderen IIC-Busteilnehmern einsetzen.

**1. Geschwindigkeitsmessung im MC-Car**

Abbildung 1 zeigt das Prinzip der Motoransteuerung und Geschwindigkeitsmessung durch die Traget MCU. Details zur Adressierung der Encoder MCU über den IIC-Bus finden Sie im Encoder_Datenblatt.pdf.




**2. IIC-Bussystem**

Legen Sie in CW ein neues C-Projekt mit Copy/Paste an und nutzen Sie das gegebene File main.c als Hauptdatei. Fügen Sie in der Bibliothek MC_Library zu den Verzeichnissen Lib_Headers bzw. Lib_Sources die gegebenen Files error.h, i2c.h und quad.h bzw. quad_temp.c und i2c_temp.c zu.

a) Analysieren Sie den gegebenen Code und implementieren Sie im File i2c_temp.c die IICInitialisierung sowie die beiden zusammengesetzten IIC-Funktionen unter Verwendung der IICGrundfunktionen (siehe CW Task-View).
   Testen Sie Ihre Implementierung in dem Sie die Räder des MC-Cars drehen (von Hand oder über die Fernbedienung) und im Debugger die Werte der globalen Variable speed beobachten.
b) Implementieren Sie dann im File quad_temp.c die beiden Funktionen zum Auslesen und Rücksetzen der Tick-Counter des Encoders.
   Testen Sie Ihre Implementierung in dem Sie die Räder des MC-Cars langsam drehen und im Debugger die Werte der globalen Variable ticks beobachten.

# i2c.c

**[…]**

```
//---------------------------------------------------------------------------------------------
// I 2 C   -   Z U S A M M E N G E S E T Z T E   F U N K T I O N E N
//---------------------------------------------------------------------------------------------


/**
 * Prüft, ob ein I2C-Device antwortet.
 *
 * @param [in] ucAdr
 *      die I2C-Adresse des gewünschten Gerätes
 *
 * @return
 *      EC_SUCCESS          falls der Slave immer mit ACK geantwortet hat
 *      EC_I2C_NO_ANSWER    falls I2C-Gerät nicht geantwortet hat
 */
tError i2cTest(uint8 adr)
{
  tError result;

  result = i2cStart(adr, FALSE);
  if (result != EC_SUCCESS) return result;

  i2cStop();
  return EC_SUCCESS;
}


 /**
 * Liest Daten von einem I2C-Gerät dass zusätzlich ein
 * Befehls-Byte benötigt.
 *
 * @param [in] adr
 *      die I2C-Adresse des gewünschten Gerätes
 * @param [in] cmd
 *      der gewünschte Befehl (Laut Datenblatt vom Slave)
 * @param [out] data
 *      die zu sendenden Daten
 * @param [in] length
 *      die Anzahl Bytes, die gesendet werden.
 *
 * @return
 *      EC_SUCCESS          falls der Slave immer mit ACK geantwortet hat
 *      EC_I2C_NAK          falls kein ACK empf. wurde.
 *      EC_I2C_NO_ANSWER    falls I2C-Gerät nicht geantwortet hat
 */
tError i2cReadCmdData(uint8 adr, uint8 cmd, uint8 *data, uint8 length)
{
  tError result;

  result = i2cStart(adr, TRUE);             // adr senden -> WRITE
  if (result != EC_SUCCESS) return result;  // bei einem Error abbrechen

  result = i2cSendData(&cmd, 1);            // cmd senden
  if (result != EC_SUCCESS) return result;  // bei einem Error abbrechen

  result = i2cRepeatedStart(adr, FALSE);    // adr senden -> READ
  if (result != EC_SUCCESS) return result;  // bei einem Error abbrechen

  i2cReceiveData(data, length);             // Daten empfangen (i2cStop wird automatisch aufgerufen)

  return EC_SUCCESS;                        // Success
}
```

```
/**
 * Sendet Daten zu einem I2C-Gerät, das zusätzlich ein
 * Befehls-Byte benötigt.
 *
 * @param [in] adr
 *      die I2C-Adresse des gewünschten Gerätes
 * @param [in] cmd
 *      der gewünschte Befehl (Laut Datenblatt vom Slave)
 * @param [in] data
 *      die zu sendenden Daten
 * @param [in] length
 *      die Anzahl Bytes, die gesendet werden.
 *
 * @return
 *      EC_SUCCESS          falls der Slave immer mit ACK geantwortet hat
 *      EC_I2C_NAK          falls kein ACK empf. wurde.
 *      EC_I2C_NO_ANSWER    falls I2C-Gerät nicht geantwortet hat
 */
tError i2cWriteCmdData(uint8 adr, uint8 cmd, uint8 *data, uint8 length)
{
  tError result;

  result = i2cStart(adr, TRUE);             // adr senden (write)
  if (result != EC_SUCCESS) return result;  // bei einem Error abbrechen

  result = i2cSendData(&cmd, 1);            // cmd senden
  if (result != EC_SUCCESS) return result;  // bei einem Error abbrechen

  result = i2cSendData(data, length);       // data senden
  if (result != EC_SUCCESS) return result;  // bei einem Error abbrechen

  i2cStop();                                // Stop i2C (free i2c Bus)
  return EC_SUCCESS;                        // Success
}


/**
 * Initialisiert den I2C-Bus
 */
void i2cInit()
{
  // enable I2C with 400 kHz SCL clock frequency
  // Bus speed = 24MHz
  // 24'000 / 400 => 60 -> 2 * 30
  IICF_MULT = 0x01;     // 0x01 mul = 02
  IICF_ICR  = 0x05;     // 0x05 icr = 30

  IICC_IICEN = 1;       // I2C einschalten
}
```

## quad.c

```
#include "platform.h"
#include "quad.h"
#include "i2c.h"


#define I2C_QUAD_ADR            0x54

#define QUAD_STATUS_CONTROL     0
#define QUAD_SPEED              1
#define QUAD_TICKS              5
#define QUAD_ERRORS             9
#define QUAD_CARRIER_MODULO    11


typedef union                           // Register 00: control/status
{
  unsigned char Byte;
  struct
  {
    tQuadMode encMode           :2;     // Bit[0-1]: Encoder Mode: 0=off, 1=Encoder, 2=Calibration
    uint8 encResetTicks         :1;     // Bit[2]: 1 = resets ticks left & right
    uint8 encResetError         :1;     // Bit[3]: 1 = resets error counter
    uint8 carrierEnable         :1;     // Bit[4]: 0=off, 1=on
    uint8 oledEnable            :1;     // Bit[5]: 0=off, 1=on
    uint8 reserved              :1;     // -
    uint8 power                 :1;     // Bit[7]: 1=power off immediately
  } Bits;
} tQuadStatusControlReg;


typedef union
{
  struct
  {
    tQuadStatusControlReg statusControl;
    int16 speedL;
    int16 speedR;
    int16 ticksL;
    int16 ticksR;
    uint8 errorL;
    uint8 errorR;
    uint16 carrierModulo;
  } tFields;

  char data[sizeof(tFields)];
} tQuadReg;



/**
 * Returns the operating mode
 *
 * @return
 *    a tQuadMode enum value.
 */
tQuadMode quadGetMode(void)
{
  tQuadStatusControlReg sc;
  tError result = i2cReadCmdData(I2C_QUAD_ADR, QUAD_STATUS_CONTROL, &sc.Byte,
sizeof(tQuadStatusControlReg));

  if (result == EC_SUCCESS) return sc.Bits.encMode;
  return qmUnkownError;
}
```

```
/**
 * Sets the operating mode:
 * - 0 => Off (power saving mode)
 * - 1 => normal operating mode
 * - 2 => calibration mode
 * - 3 => reserved
 *
 * @param [in] mode
 *    a pointer to a tQuadMode structure.
 *
 * @return
 *    EC_SUCCESS if the command was executed sucessfully
 *    EC_I2C_NAK if the device answered with an NAK
 *    EC_I2C_NO_ANSWER if the device did not answer (wrong address?)
 */
tError quadSetMode(tQuadMode mode)
{
  tQuadStatusControlReg sc;
  tError result = i2cReadCmdData(I2C_QUAD_ADR, QUAD_STATUS_CONTROL, &sc.Byte,
sizeof(tQuadStatusControlReg));

  if (result == EC_SUCCESS)
  {
    sc.Bits.encMode = (uint8)mode;
    result = i2cWriteCmdData(I2C_QUAD_ADR, QUAD_STATUS_CONTROL, (char*)&sc,
sizeof(tQuadStatusControlReg));
  }
  return result;
}


/**
 * Returns the speed of the left and right wheel.
 * The Speed is in mm/sec.
 *
 * @param [out] speed
 *    a pointer to a tQuadSpeed structure that includes both speed values
 *
 * @return
 *    EC_SUCCESS if the command was executed sucessfully
 *    EC_I2C_NAK if the device answered with an NAK
 *    EC_I2C_NO_ANSWER if the device did not answer (wrong address?)
 */
tError quadGetSpeed(pQuadSpeed speed)
{
  return i2cReadCmdData(I2C_QUAD_ADR, QUAD_SPEED, (char*)speed, sizeof(tQuadSpeed));
}


/**
 * Returns both ticks counter (left and right)
 *
 * @param [out] ticks
 *    a pointer to a tQuadTicks structure that includes both counters
 *
 * @return
 *    EC_SUCCESS if the command was executed sucessfully
 *    EC_I2C_NAK if the device answered with an NAK
 *    EC_I2C_NO_ANSWER if the device did not answer (wrong address?)
 */
tError quadGetTicks(pQuadTicks ticks)
{
  return i2cReadCmdData(I2C_QUAD_ADR, QUAD_TICKS, (char*)ticks, sizeof(tQuadTicks));
}
```

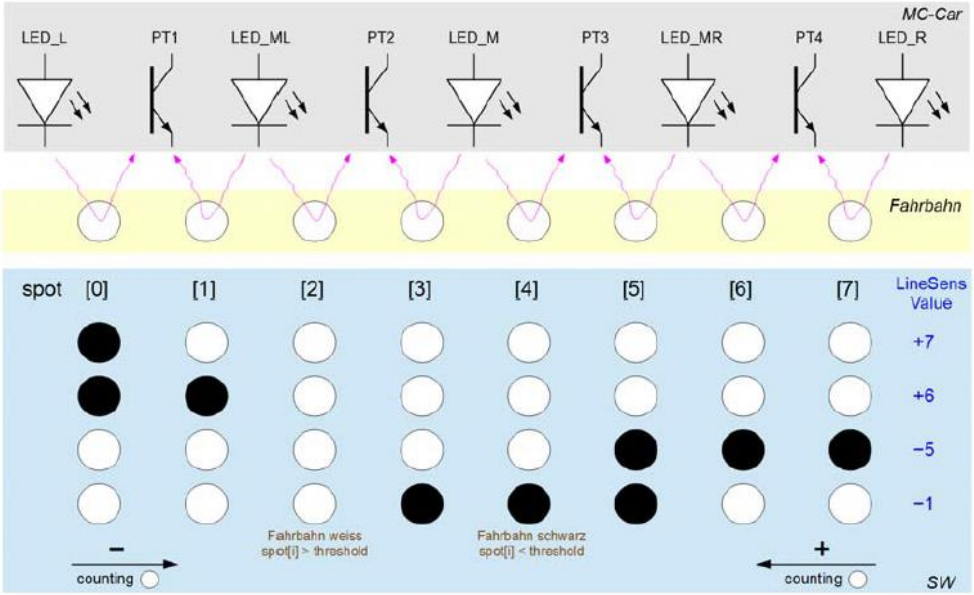
```
/**
 * Resets both ticks counter to zero (left and right).
 *
 * @return
 *    EC_SUCCESS if the command was executed sucessfully
 *    EC_I2C_NAK if the device answered with an NAK
 *    EC_I2C_NO_ANSWER if the device did not answer (wrong address?)
 */
tError quadResetTicks(void)
{
  tQuadStatusControlReg sc;

  tError result = i2cReadCmdData(I2C_QUAD_ADR, QUAD_STATUS_CONTROL, &sc.Byte,
sizeof(tQuadStatusControlReg));

  if (result == EC_SUCCESS)
  {
    sc.Bits.encResetTicks = TRUE;
    result = i2cWriteCmdData(I2C_QUAD_ADR, QUAD_STATUS_CONTROL, (char*)&sc,
sizeof(tQuadStatusControlReg));
  }

  return result;
}


/**
 * Returns both ticks error counter (left and right)
 *
 * @param [out] errors
 *    a pointer to a tQuadErrors structure that includes both error counters
 *
 * @return
 *    EC_SUCCESS if the command was executed sucessfully
 *    EC_I2C_NAK if the device answered with an NAK
 *    EC_I2C_NO_ANSWER if the device did not answer (wrong address?)
 */
tError quadGetErrors(pQuadErrors errors)
{
  return i2cReadCmdData(I2C_QUAD_ADR, QUAD_ERRORS, (char*)errors, sizeof(tQuadErrors));
}


/**
 * Resets both ticks error counter to zero.
 *
 * @return
 *    EC_SUCCESS if the command was executed sucessfully
 *    EC_I2C_NAK if the device answered with an NAK
 *    EC_I2C_NO_ANSWER if the device did not answer (wrong address?)
 */
tError quadResetErrors(void)
{
  tQuadStatusControlReg sc;

  tError result = i2cReadCmdData(I2C_QUAD_ADR, QUAD_STATUS_CONTROL, &sc.Byte,
sizeof(tQuadStatusControlReg));

  if (result == EC_SUCCESS)
  {
    sc.Bits.encResetError = TRUE;
    result = i2cWriteCmdData(I2C_QUAD_ADR, QUAD_STATUS_CONTROL, &sc.Byte,
sizeof(tQuadStatusControlReg));
  }

  return result;
}
```

# 9: Analog-Digital Wandler

Sie verstehen die AD-Wandlung nach dem Prinzip der sukzessiven Approximation und können das AD-Wandlersystem im MC9S08JM60 zur Auswertung von analogen Signalen einsetzen

### 1. Liniensensoren des MC-Cars

Nachfolgend ist die geometrische Anordnung der 5 Infrarot-LEDs und 4 Phototransistoren des MC-Cars sowie der in SW implementierte Auswerte-Algorithmus zur Linienerkennung dargestellt.




### 2. AD-Wandlung

Legen Sie in CW ein neues C-Projekt mit Copy/Paste an und nutzen Sie das gegebene File main.c als Hauptdatei. Fügen Sie in der Bibliothek MC_Library zu den Verzeichnissen Lib_Headers bzw. Lib_Sources die gegebenen Files linesens.h und adc.h bzw. linesens.c zu (Drag/Drop/Copy).

Implementieren Sie im File adc_temp.c an den 4 gekennzeichneten Stellen (siehe CW Task-View) die fehlende Funktionalität, so dass das Auto der gegebenen Testlinie folgen kann.

## adc.c

```
#include "platform.h"
#include "adc.h"


#define ADC_RES_8BIT      0
#define ADC_RES_10BIT     2
#define ADC_RES_12BIT     1


/**
 * Performs one A/D conversion for the specified channel
 * with 12 bit resolution. The function blocks until the
 * conversion has been finished.
 */
uint16 adcGet12BitValue(AdcChannels ch)
{
  ADCCFG_MODE = ADC_RES_12BIT;  // 12-bit conversion
  ADCSC1_ADCH = ch;             // select channel --> start conversion
  while (ADCSC1_COCO == 0) {}   // wait until conversion complete
  return ADCR;                  // converted value -> return result
}

/**
 * Performs one A/D conversion for the specified channel
 * with 10 bit resolution. The function blocks until the
 * conversion has been finished.
 */
uint16 adcGet10BitValue(AdcChannels ch)
{
  ADCCFG_MODE = ADC_RES_10BIT;  // 10-bit conversion
  ADCSC1_ADCH = ch;             // select channel --> start conversion
  while (ADCSC1_COCO == 0) {}   // wait until conversion complete
  return ADCR;                  // converted value -> return result
}

/**
 * Performs one A/D conversion for the specified channel
 * with 8 bit resolution. The function blocks until the
 * conversion has been finished.
 */
uint8 adcGet8BitValue(AdcChannels ch)
{
  ADCCFG_MODE = ADC_RES_8BIT;   // 8-bit conversion
  ADCSC1_ADCH = ch;             // select channel --> start conversion
  while (ADCSC1_COCO == 0) {}   // wait until conversion complete
  return ADCRL;                 // converted value -> return result
}


/**
 * Configures the A/D converter as follows:
 * - high speed mode with long sample time
 * - set ADCK = 6 MHz (bus clock / 4)
 * - enables the four line sensors
 */
void adcInit(void)
{
  APCTL1 = 0xF0;        // Enabled Pin 4..7 (line sensors)

  ADCCFG_ADLPC = 0;     // High speed Mode
  ADCCFG_ADLSMP = 1;    // long sample time
  ADCCFG_ADIV = 2;      // clock divide -> 4
  ADCCFG_ADICLK = 0;    // input clock -> Bus Clock
}
```

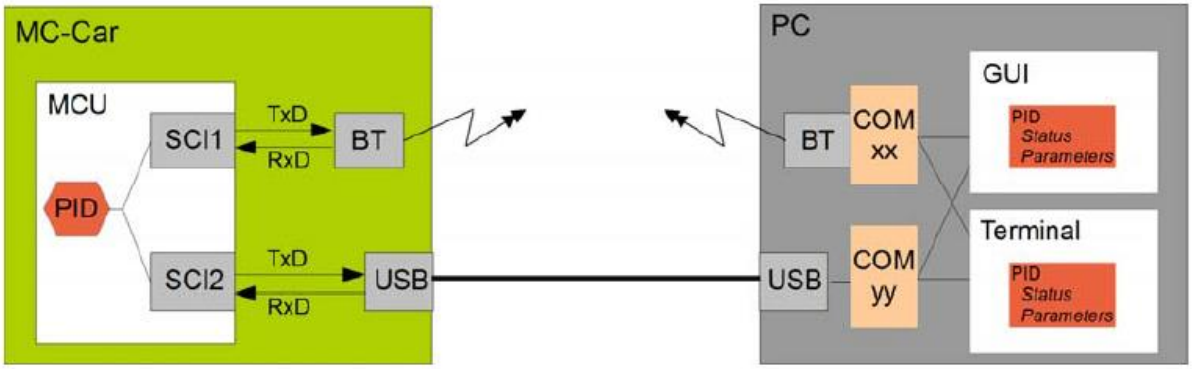
# 10: Serial Communication Interface (SCI)

Sie verstehen das RS-232 Protokoll sowie das Zusammenspiel zwischen HW und SW bei der Kommunikation über die serielle Schnittstelle.

### 1. Systemüberblick

Vom PC aus sollen im MC-Car die Parameter des PID-Geschwindigkeitsreglers eingestellt sowie der aktuelle Status (Sollwert, Istwert, Regelabweichung etc.) gelesen werden können.

Nachfolgend ist der Systemaufbau für die serielle Kommunikation zwischen MC-Car und PC gezeigt. Die beiden möglichen physischen Verbindungen (USB, Bluetooth) sowie die beiden möglichen Benutzerschnittstellen (Terminal, GUI) sind in den folgenden Kapiteln beschrieben.



## 2. Physische Verbindungen

### a) Verbindung über USB

Das Debug-Interface auf dem MC-Car beinhaltet zusätzlich einen USB-zu-RS232 Wandler. Im Geräte-Manager (unter Systemsteuerung => System) sollten Sie den entsprechenden Com-Port ausfindig machen können. Es wird kein zusätzlicher Treiber benötigt.

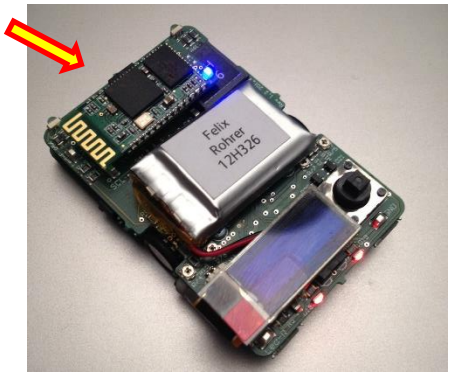
Um über USB zu kommunizieren, stellen Sie in sci.h die Konfiguration um auf SCI2:

```
#ifndef SCI_H
#define SCI_H

#include "platform.h"

// 1 = Bluetooth, 2 = Cable
#define SCI_DEFAULT                2       // [0|1|2]  0=>aus, 1=>SCI1, 2=>SCI2
```

### b) Verbindung über Bluetooth

Bei dieser Variante müssen sie zuerst das Bluetooth-Modul bei ausgeschaltetem MC-Car aufsetzen. Achten sie dabei unbedingt auf die Polarität und die richtige Positionierung der Steckerreihen übereinander.

**Achtung: Das Bluetooth-Modul kann durch falsches Aufstecken zerstört werden!**




Wählen Sie als nächstes die Seriennummer als Namen für das Bluetooth Modul und tragen Sie diese in platform.h ein:

```
#define BLUETOOTH_NAME          "12H301"
```

Um über Bluetooth zu kommunizieren, stellen Sie in sci.h die Konfiguration um auf SCI1:

```
#ifndef SCI_H
#define SCI_H

#include "platform.h"

// 1 = Bluetooth, 2 = Cable
#define SCI_DEFAULT                1       // [0|1|2]  0=>aus, 1=>SCI1, 2=>SCI2
```

Stellen Sie nun wie folgt eine Verbindung zum Bluetooth Modul her:

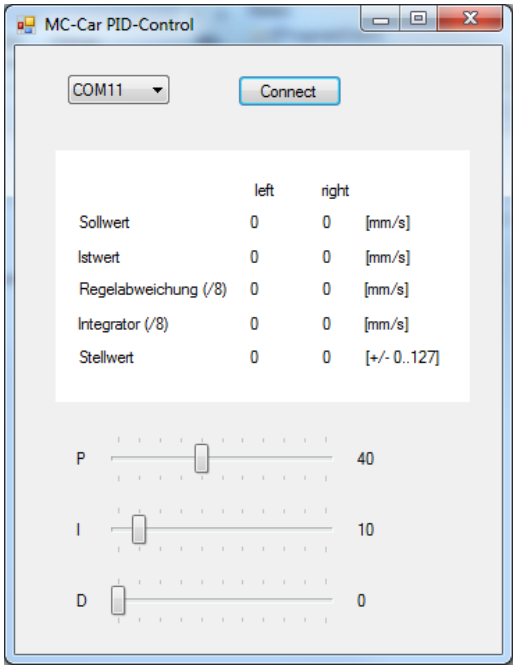
(Win7) Start => Geräte und Drucker => Geräte hinzufügen

Hier sollten Sie Ihren MC-Car sehen. Das Passwort zum Verbindungsaufbau lautet: „**1234**“

### 3. Benutzerschnittstellen

a) GUI

Starten Sie die Applikation MCCar.exe und verbinden Sie sich über den entsprechenden COM-Port mit dem MC-Car.



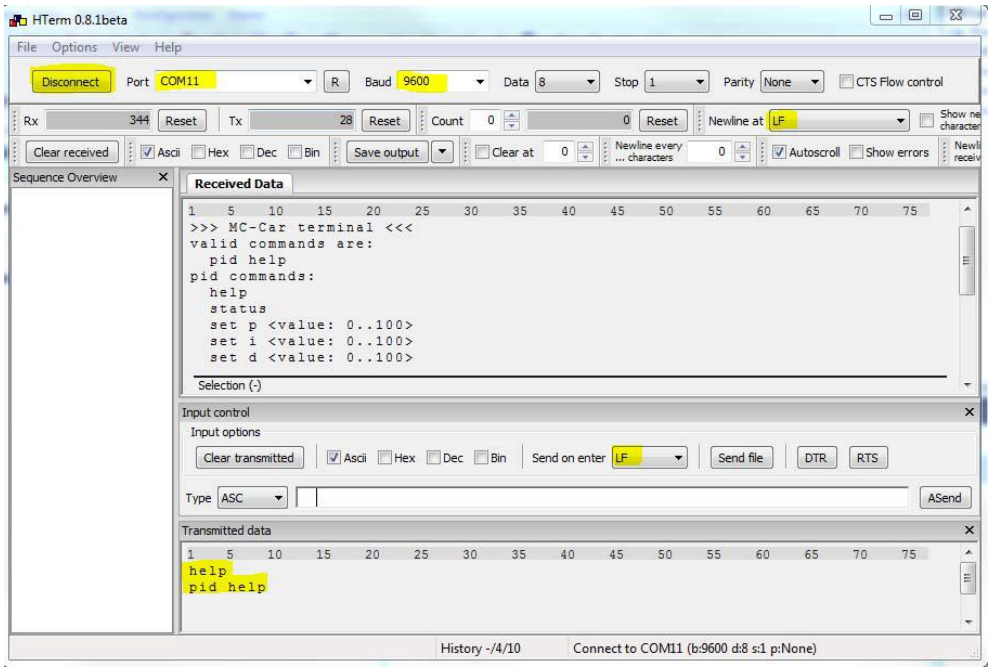

**b) Terminalprogramm**

Nehmen Sie im Terminalprogramm (z.B. HTerm) die unten gezeigten Einstellungen vor:

Baudrate = 9600, Format = 8-None-1, Newline at = LF, Send on Enter = LF

und verbinden Sie sich über den entsprechenden COM-Port mit dem MC-Car.

Nachdem Senden des Kommandos „help“ vom Terminalprogramm, werden die verfügbaren Kommandos angezeigt, siehe Implementierung in term.c.



### 4. RS-232 Software für MC-Car

Legen Sie in CW ein neues C-Projekt mit Copy/Paste an und verwenden Sie die gegebenen Files main.c und Project.prm für das neue Projekt. Aktualisieren Sie in Ihrer Bibliothek MC_Library die Verzeichnisse Lib_Headers bzw. Lib_Sources mit den gegebenen Bibliotheksdateien.

Analysieren Sie den gegebenen Code und implementieren Sie im File sci1_temp.c oder/und sci2_temp.c (je nachdem ob Sie mit Bluetooth oder/und USB-Verbindung arbeiten) die folgenden Funktionalitäten (siehe CW Task-View):

a) in sci1/2Init() eine Baud Rate von 9600 Baud/s unter Verwendung des define-Wertes BUSCLOCK.
   Stellen Sie dabei sicher, dass der zu konfigurierende SBR -Wert zum nächsten ganzzahligen Wert gerundet wird.
   Tipp: x + 0.5 = (10 x + 5) / 10
b) in der ISR sci1/2RxD() das Auslesen der empfangenen Bytes aus dem SCI Data Register und Schreiben in den Rx-Ringbuffer. Danach sollten die Reglerparameter z.B. über das GUI im MC-Car verändert werden können.
c) in der ISR sciXTxD() das Auslesen der zusendenden Bytes aus dem Tx-Rinbuffer und Schreiben ins SCI Data Register. Danach sollte ebenfalls das Auslesen des Regler-Status funktionieren.

## sci1.c

**[…]**

```
/**
 * RxD Interrupt-Routine
 *
 * Liest das empfangene Byte und speichert es in der Empfangsqueue.
 *
 * @details
 * Um den Empfangsinterrupt zu quittieren muss zuerst da Statusregister SCIxS1 gelesen
 * werden und anschliessend muss das empfange Zeichen aus SCIxD gelesen werden.
 */
interrupt void sci1RxD(void)
{
  char ch;
  (void)SCI1S1;       // 1. Status lesen
  ch = SCI1D;         // 2. empf. Zeichen lesen => Interrupt wird quittiert.

  // @ToDo write received data byte into the rx buffer
  if (rx1BufCount < SCI1_RX_BUF_SIZE)
  {
    rx1Buf[rx1BufWritePos] = ch;
    rx1BufCount++;
    rx1BufWritePos++;
    if (rx1BufWritePos == SCI1_RX_BUF_SIZE) rx1BufWritePos = 0;
  }
}


/**
 * TxD Interrupt-Routine
 *
 * Holt das nächste Byte aus der Queue und versendet es.
 * Falls die Queue leer ist, dann wird der Interrupt deaktiviert.
 */
interrupt void sci1TxD()
{
  (void)SCI1S1;       // 1. Status lesen

  // @ToDo read from the tx buffer and write the data into the sci data register.
  // Disable tx interrupt if tx buffer is empty
  if (tx1BufCount > 0)
  {
    SCI1D = tx1Buf[tx1BufReadPos];
    tx1BufCount--;
    tx1BufReadPos++;
    if (tx1BufReadPos == SCI1_TX_BUF_SIZE) tx1BufReadPos = 0;
  } else {
    // Buffer leer => Transmit Data Register Empty Interrupt deaktivieren
    // wird später durch sciWriteByte() wieder aktiviert...
    SCI1C2_TIE = 0;
  }
}
```

**[…]**

## sci2.c

**[…]**

```
/**
 * RxD Interrupt-Routine
 *
 * Liest das empfangene Byte und speichert es in der Empfangsqueue.
 *
 * @details
 * Um den Empfangsinterrupt zu quittieren muss zuerst da Statusregister SCIxS1 gelesen
 * werden und anschliessend muss das empfange Zeichen aus SCIxD gelesen werden.
 */
interrupt void sci2RxD(void)
{
  char ch;
  (void)SCI2S1;            // 1. Status lesen
  ch = SCI2D;        // 2. empf. Zeichen lesen => Interrupt wird quittiert.

  // @ToDo write received data byte into the rx buffer
  if (rx2BufCount < SCI2_RX_BUF_SIZE)
  {
    rx2Buf[rx2BufWritePos] = ch;
    rx2BufCount++;
    rx2BufWritePos++;
    if (rx2BufWritePos == SCI2_RX_BUF_SIZE) rx2BufWritePos = 0;
  }
}


/**
 * TxD Interrupt-Routine
 *
 * Holt das nächste Byte aus der Queue und versendet es.
 * Falls die Queue leer ist, dann wird der Interrupt deaktiviert.
 */
interrupt void sci2TxD()
{
  (void)SCI2S1;       // 1. Status lesen

  // @ToDo read from the tx buffer and write the data into the sci data register.
  // Disable tx interrupt if tx buffer is empty
  if (tx2BufCount > 0)
  {
    SCI2D = tx2Buf[tx2BufReadPos];
    tx2BufCount--;
    tx2BufReadPos++;
    if (tx2BufReadPos == SCI2_TX_BUF_SIZE) tx2BufReadPos = 0;

  } else {
    // Buffer leer => Transmit Data Register Empty Interrupt deaktivieren
    // wird später durch sciWriteByte() wieder aktiviert...
    SCI2C2_TIE = 0;
  }
}
```

**[…]**

# 11: Echtzeitbetriebssystem uCOS-II

Sie lernen anhand aufeinander aufbauender Übungen, welche Möglichkeiten ein Echtzeitbetriebssystem wie das uCosII bietet und wie sich damit MC-Applikationen realisieren lassen.

### Aufgabe 1: Blinklicht

Implementieren Sie einen Task, welcher die LED 1 von Port D (PTDD0) und einer, der die LED 3 von Port D (PTDD3) blinken lässt. Damit die Unabhängigkeit der Tasks sichtbar wird, lassen Sie die beiden LEDs unterschiedlich schnell blinken.

**aufgabe1.c**

```
#include <hidef.h>      // for EnableInterrupts macro
#include "platform.h"   // include peripheral declarations
#if AUFGABE == 1

#define PRIO_INIT           0           // max. priority for the init task.
#define PRIO_BLINK_D1       1
#define PRIO_BLINK_D3       2

#define TASK_STK_SIZE       128         // Stacksize for one Task

OS_STK InitTaskStack[TASK_STK_SIZE];    // allocate memory for the stack of the init task
OS_STK BlinkD1Stack[TASK_STK_SIZE];     // allocate memory for the stack of the BlinkD1 task
OS_STK BlinkD3Stack[TASK_STK_SIZE];     // allocate memory for the stack of the BlinkD3 task


/**
 * This Task lets Bit1 of Port D blinking.
 *
 * @param pdata optional data (unused).
 */
void BlinkD1(void *pdata)
{
    int i = 0;

    (void)pdata;                        // prevents compiler warning
    for(;;)
    {
      // @todo implement BlinkD1-Task here
      PTDD_PTDD1 = 0;
      OSTimeDly(50); // 50 * 20ms = 1'000ms delay

      PTDD_PTDD1 = 1;
      OSTimeDly(50); // 50 * 20ms = 1'000ms delay
    }
}


/**
 * This task lets Bit3 of Port D blinking.
 *
 *  @param pdata optional data (unused).
 */
void BlinkD3(void *pdata)
{
    (void)pdata;

    for(;;)
    {
      // @todo implement BlinkD3-Task here...
      PTDD_PTDD3 = 0;
      (void)OSTimeDlyHMSM(0,0,0,50);

      PTDD_PTDD3 = 1;
      (void)OSTimeDlyHMSM(0,0,0,50);
    }
}
```

```
/**
 * Timer etc. initialisieren...
 *
 * @param pdata Optionale Daten für den Task beim Start.
 */
void InitTask(void* pdata)
{
    PTDDD_PTDDD1 = 1; // Output
    PTDDD_PTDDD3 = 1; // Output
    PTDD_PTDD1 = 0;   // Enable LED
    PTDD_PTDD3 = 0;   // Enable LED
    // PTDD = 0xFF;   // alle LED Off
    // PTDDD = 0xFF;  // set Output

    (void)pdata;
    OSTimerInit();                  // start timer (RTI)

    (void)OSTaskCreate(BlinkD1, (void*)0, &BlinkD1Stack[TASK_STK_SIZE-1], PRIO_BLINK_D1);
    (void)OSTaskCreate(BlinkD3, (void*)0, &BlinkD3Stack[TASK_STK_SIZE-1], PRIO_BLINK_D3);

    (void)OSTaskDel(OS_PRIO_SELF);  // destroy the init task.
}


/**
 * main program
 *
 * This function has the following tasks:
 * - init system clock
 * - init operating system
 * - create the first (init-) task
 * - start the os
 */
void main(void)
{
    OSInit();                       // initialize the operating system

    // create the init task
    (void)OSTaskCreate(InitTask, (void*)0, &InitTaskStack[TASK_STK_SIZE-1], PRIO_INIT);

    OSStart();                      // start the operating system
}
#endif
```

**Aufgabe 2: Zwei parallele Lauflichter**

In dieser Aufgabe geht es darum, zwei unabhängige Lauflichter zu implementieren. Sobald ein Lauflicht seine Randstelle erreicht, soll es die Richtung wechseln. Beachten Sie dazu, ausgehend von Aufgabe 1, folgendes:

- Verwenden Sie für das erste Lauflicht die LEDs PTDD0..3 und für das zweite Lauflicht die LEDs PTDD4..7
- In einem ersten Schritt wird die Schrittgeschwindigkeit als fixer Wert implementiert.
- Implementieren Sie das Multitasking-Programm mit Hilfe von 3 Tasks. Einen Task für die Initialisierung, sowie je einen Task für die Lauflichter.
- Testen Sie das Programm
- Implementieren Sie mit Hilfe eines weiteren Tasks die Geschwindigkeitsregelung mit Eingabe via den Potentiometern auf dem L-Print. PTB6 stellt die Geschwindigkeit für das linke Lauflicht (PTDD4..7) und PTB7 für das rechte Lauflicht (PTDD0..3) ein.

## aufgabe2.c

```
#include <hidef.h>      // for EnableInterrupts macro
#include "platform.h"   // include peripheral declarations
#if AUFGABE == 2

#define PRIO_INIT           0           // max. priority for the init task.
#define PRIO_ADC            1
#define PRIO_LAUFL_1        2
#define PRIO_LAUFL_2        3

#define TASK_STK_SIZE       128         // Stacksize for one Task

OS_STK InitTaskStack[TASK_STK_SIZE];    // allocate memory for the stack of the init task
OS_STK ADCStack[TASK_STK_SIZE];         // allocate memory for the stack of the BlinkD1 task
OS_STK LAUFL1Stack[TASK_STK_SIZE];      // allocate memory for the stack of the BlinkD1 task
OS_STK LAUFL2Stack[TASK_STK_SIZE];      // allocate memory for the stack of the BlinkD3 task

// init ADC global var
unsigned char ucDelayKitRight = 10;
unsigned char ucDelayKitLeft  = 10;

/**
 * This task reads alternately the value of ATDCH 6 and 7 and
 * stores the 8 bit value in the variables ucDelayKitLeft and
 * ucDelayKitRight.
 *
 * @param pdata optional data (unused).
 */
void AdcTask(void *pdata)
{
    (void)pdata;
    for(;;)
    {
        OSTimeDly(5);
        if(ATD1SC_ATDCH == 7)
        {                                       // if +1 is neglected KitLeft
            ucDelayKitRight = (ATD1RH / 4) + 1; // is blocked when poti7 is in
            ATD1SC_ATDCH=6;                     // left most position, because
        }                                       // KitLeft has lower prio
        else
        {
            ucDelayKitLeft = (ATD1RH / 4) + 1;
            ATD1SC_ATDCH=7;
        }
    }
}


/**
 * Timer etc. initialisieren...
 *
 * @param pdata Optionale Daten für den Task beim Start.
 */
void adcInit(void)
{
  ATD1C_ATDPU=1;                // ATD power on
  ATD1C_PRS = 2;                // set the prescaler. Valid for a busclock of 3-12 MHz.
  ATD1C_RES8=1;                 // 8BitData
  ATD1PE_ATDPE7=1;              // Pin enabled Channel 7
  ATD1PE_ATDPE6=1;              // Pin enabled Channel 6
  ATD1SC_ATDCO=0;               // continuous conversion
  ATD1SC_ATDCH=7;               // Select Input Channel 7
}
```

```
/**
 * Lauflicht PTDD0..3
 *
 * @param pdata optional data (unused).
 */
void Lauflicht1(void *pdata)
{
    unsigned char ucBitPos = 1;
    unsigned char ucGoLeft = 1;

    (void)pdata;                          // prevents compiler warning
    for(;;)
    {
      switch(ucBitPos)
      {
        case 1: PTDD_PTDD0 = 1; break;
        case 2: PTDD_PTDD1 = 1; break;
        case 4: PTDD_PTDD2 = 1; break;
        case 8: PTDD_PTDD3 = 1; break;
      }

      if (ucGoLeft) {
        ucBitPos <<= 1;
        if (ucBitPos == 8) ucGoLeft = 0;
      } else {
        ucBitPos >>= 1;
        if (ucBitPos == 1) ucGoLeft = 1;
      }

      switch(ucBitPos)
      {
        case 1: PTDD_PTDD0 = 0; break;
        case 2: PTDD_PTDD1 = 0; break;
        case 4: PTDD_PTDD2 = 0; break;
        case 8: PTDD_PTDD3 = 0; break;
      }

      OSTimeDly(ucDelayKitRight); // 50 * 20ms = 1'000ms delay
    }
}

/**
 * Lauflicht PTDD4..8
 *
 * @param pdata optional data (unused).
 */
void Lauflicht2(void *pdata)
{
    unsigned char ucBitPos = 1;
    unsigned char ucGoLeft = 1;

    (void)pdata;                          // prevents compiler warning
    for(;;)
    {
      switch(ucBitPos)
      {
        case 1: PTDD_PTDD4 = 1; break;
        case 2: PTDD_PTDD5 = 1; break;
        case 4: PTDD_PTDD6 = 1; break;
        case 8: PTDD_PTDD7 = 1; break;
      }

      if (ucGoLeft) {
        ucBitPos <<= 1;
        if (ucBitPos == 8) ucGoLeft = 0;
      } else {
        ucBitPos >>= 1;
        if (ucBitPos == 1) ucGoLeft = 1;
      }

      switch(ucBitPos)
      {
        case 1: PTDD_PTDD4 = 0; break;
        case 2: PTDD_PTDD5 = 0; break;
        case 4: PTDD_PTDD6 = 0; break;
        case 8: PTDD_PTDD7 = 0; break;
      }

      OSTimeDly(ucDelayKitLeft); // 50 * 20ms = 1'000ms delay
    }
}
```

```
/**
 * Timer etc. initialisieren...
 *
 * @param pdata Optionale Daten für den Task beim Start.
 */
void InitTask(void* pdata)
{
    PTDD = 0xFF;   // LED off

    PTDDD = 0xFF;  // set Output

    (void)pdata;
    OSTimerInit();                  // start timer (RTI)

    adcInit();  // init ADC


   (void)OSTaskCreate(AdcTask, (void*)0, &ADCStack[TASK_STK_SIZE-1], PRIO_ADC);
   (void)OSTaskCreate(Lauflicht1, (void*)0, &LAUFL1Stack[TASK_STK_SIZE-1], PRIO_LAUFL_1);
   (void)OSTaskCreate(Lauflicht2, (void*)0, &LAUFL2Stack[TASK_STK_SIZE-1], PRIO_LAUFL_2);

    (void)OSTaskDel(OS_PRIO_SELF);  // destroy the init task.
}


/**
 * main program
 *
 * This function has the following tasks:
 * - init system clock
 * - init operating system
 * - create the first (init-) task
 * - start the os
 */
void main(void)
{
    OSInit();                       // initialize the operating system

    // create the init task
    (void)OSTaskCreate(InitTask, (void*)0, &InitTaskStack[TASK_STK_SIZE-1], PRIO_INIT);

    OSStart();                      // start the operating system
}
#endif
```

**Aufgabe 3: "Rendezvous" der Lauflichter (Semaphore)**

Die zwei unabhängigen, „parallel“ laufenden Tasks aus Aufgabe 2 sollen sich jeweils beim MSB (also bei PTDD7 bzw. PTDD3) synchronisieren. Damit dieses Rendezvous stattfinden kann, muss das schnellere Lauflicht auf das langsamere Lauflicht warten. Realisieren Sie diese Kooperation mit Semaphoren.

Vorgehen:

- Verschaffen Sie sich einen Überblick über die Funktionsweise und den Einsatz von Semaphoren.
- Implementieren Sie das Rendezvous mit Hilfe von Semaphoren und testen es aus.

## aufgabe3.c

```
#include <hidef.h>      // for EnableInterrupts macro
#include "platform.h"   // include peripheral declarations
#if AUFGABE == 3

#define PRIO_INIT           0           // max. priority for the init task.
#define PRIO_ADC            1           // high priority for the adc task.
#define PRIO_KIT_RIGHT      2
#define PRIO_KIT_LEFT       3

#define TASK_STK_SIZE       128         // Stacksize for one Task


OS_STK InitTaskStack[TASK_STK_SIZE];    // allocate memory for the stack of the init task
OS_STK KitRightStack[TASK_STK_SIZE];    // allocate memory for the stack of the KitRight task
OS_STK KitLeftStack[TASK_STK_SIZE];     // allocate memory for the stack of the KitLeft task
OS_STK AdcStack[TASK_STK_SIZE];         // allocate memory for the stack of the ADC task

OS_EVENT *semKitLeft;                   // Semaphor
OS_EVENT *semKitRight;                  // Semaphor

unsigned char ucDelayKitLeft = 10;
unsigned char ucDelayKitRight = 10;


/**
 * This task reads alternately the value of ATDCH 6 and 7 and
 * stores the 8 bit value in the variables ucDelayKitLeft and
 * ucDelayKitRight.
 *
 * @param pdata optional data (unused).
 */
void AdcTask(void *pdata)
{
    (void)pdata;
    for(;;)
    {
        OSTimeDly(5);
        if(ATD1SC_ATDCH == 7)
        {
            ucDelayKitRight = (ATD1RH / 4) + 1;
            ATD1SC_ATDCH=6;
        }
        else
        {
            ucDelayKitLeft = (ATD1RH / 4) + 1;
            ATD1SC_ATDCH=7;
        }
    }
}
```

```
/**
 * Kit Right (bit D0..D3)
 *
 * @param pdata optional data (unused).
 */
void KitRightTask(void *pdata)
{
    unsigned char ucBitPos = 1;
    unsigned char ucGoLeft = 1;
    unsigned char ucResult;
    (void)pdata;                        // prevents compiler warning

    for (;;)
    {
        switch (ucBitPos)
        {
            case 1: PTDD_PTDD0 = 1; break;
            case 2: PTDD_PTDD1 = 1; break;
            case 4: PTDD_PTDD2 = 1; break;
            case 8: PTDD_PTDD3 = 1; break;
        }

        if (ucGoLeft)
        {
            ucBitPos <<= 1;
            if (ucBitPos == 8) ucGoLeft = 0;
        }
        else
        {
            ucBitPos >>= 1;
            if (ucBitPos == 1) ucGoLeft = 1;
        }

        switch (ucBitPos)
        {
            case 1: PTDD_PTDD0 = 0; break;
            case 2: PTDD_PTDD1 = 0; break;
            case 4: PTDD_PTDD2 = 0; break;
            case 8:
                PTDD PTDD3 = 0;
                (void)OSSemPost(semKitLeft);
                OSSemPend(semKitRight, 0, &ucResult);
                break;
        }
        OSTimeDly(ucDelayKitRight);
    }
}
```

```
/**
 * Kit Left (bit D4..D7)
 *
 *  @param pdata optional data (unused).
 */
void KitLeftTask(void *pdata)
{
    unsigned char ucBitPos = 1;
    unsigned char ucGoLeft = 1;
    unsigned char ucResult;
    (void)pdata;                        // prevents compiler warning

    for (;;)
    {
        switch (ucBitPos)
        {
            case 1: PTDD_PTDD4 = 1; break;
            case 2: PTDD_PTDD5 = 1; break;
            case 4: PTDD_PTDD6 = 1; break;
            case 8: PTDD_PTDD7 = 1; break;
        }

        if (ucGoLeft)
        {
            ucBitPos <<= 1;
            if (ucBitPos == 8) ucGoLeft = 0;
        }
        else
        {
            ucBitPos >>= 1;
            if (ucBitPos == 1) ucGoLeft = 1;
        }

        switch (ucBitPos)
        {
            case 1: PTDD_PTDD4 = 0; break;
            case 2: PTDD_PTDD5 = 0; break;
            case 4: PTDD_PTDD6 = 0; break;
            case 8:
                PTDD PTDD7 = 0;
                (void)OSSemPost(semKitRight);
                OSSemPend(semKitLeft, 0, &ucResult);
                break;
        }
        OSTimeDly(ucDelayKitLeft);
    }
}


/**
 * Timer etc. initialisieren...
 *
 * @param pdata Optionale Daten für den Task beim Start.
 */
void InitTask(void* pdata)
{
    (void)pdata;
    OSTimerInit();                  // start timer (RTI)
    PTDD = 0xFF;                    // disable LED's on Port D
    PTDDD = 0xFF;                   // configure Port D as output

    ATD1C_PRS = 2;                  // set the prescaler. Valid for a busclock of 3-12 MHz.
    ATD1C_ATDPU=1;                  // ATD power on
    ATD1C_RES8=1;                   // 8BitData
    ATD1PE_ATDPE7=1;                // Pin enabled Channel 7
    ATD1PE_ATDPE6=1;                // Pin enabled Channel 6
    ATD1SC_ATDCO=0;                 // continuous conversion
    ATD1SC_ATDCH=7;                 // Select Input Channel 7

    semKitLeft=OSSemCreate(0);
    semKitRight=OSSemCreate(0);

    (void)OSTaskCreate(AdcTask, (void*)0, &AdcStack[TASK_STK_SIZE-1], PRIO_ADC);
    (void)OSTaskCreate(KitRightTask, (void*)0, &KitRightStack[TASK_STK_SIZE-1], PRIO_KIT_RIGHT);
    (void)OSTaskCreate(KitLeftTask, (void*)0, &KitLeftStack[TASK_STK_SIZE-1], PRIO_KIT_LEFT);

    (void)OSTaskDel(OS_PRIO_SELF);  // destroy the init task.
}
```

```
/**
 * main program
 *
 * This function has the following tasks:
 * - init system clock
 * - init operating system
 * - create the first (init-) task
 * - start the os
 */
void main(void)
{
    OSInit();                       // initialize the operating system

    // create the init task
    (void)OSTaskCreate(InitTask, (void*)0, &InitTaskStack[TASK_STK_SIZE-1], PRIO_INIT);

    OSStart();                      // start the operating system
}
#endif
```

**Aufgabe 4: " Producer-Consumer" der Lauflichter**

In dieser Aufgabe soll nun das linke Lauflicht zum Producer und das rechte Lauflicht zum Consumer werden: Der Producer produziert Daten, d.h. bei jedem Schritt gibt er die Werte der Portbits PTDD4..7 zur Verarbeitung an den Consumer weiter. Die Daten werden über eine Queue, die maximal 4 Messages aufnehmen kann, an den Consumer-Task übermittelt. Der Consumer liest die Daten aus der Queue und zeigt sie an den Portbits PTD0..3 an. Falls keine Daten in der Queue vorhanden sind, so wartet der Consumer-Task bis dies der Fall ist. Falls die Queue voll ist, wartet der Producer-Task, bis wieder Platz für mindestens eine Message vorhanden ist.

Ein fünfter Task "ShowMailStore" zeigt den Füllgrad der Queue in Form einer Balkenanzeige am Port F an. Der dazu benötigte Code steht nachfolgend zur Verfügung:

```
/**
 * Shows the level (0..4) at port F as a bar graph
 *
 * @param pdata optional data (unused).
 */
void ShowMsgQueue(void *pdata)
{
    INT8U count;
    OS_Q_DATA tempData;
    (void)pdata;          // prevents compiler warning

    PTFDD = 0x0F;

    for(;;)
    {
        (void)OSQQuery(msgQueue, &tempData);
        count = (INT8U)tempData.OSNMsgs;
        PTFD = (0x0f << count);
        OSTimeDly(4);
    }
}
```

Implementieren Sie das Producer-Consumer Problem und fügen Sie den fünften Task "ShowMailStore" hinzu.

Anmerkung: Normalerweise wird bei der Queue nur ein Pointer auf die eigentliche Message übermittelt. In unserem Fall verwenden wir diesen 16-Bit Speicherplatz direkt als Datenwert und verwenden ihn nicht als Pointer. Im uCOS-II Handbuch steht dazu:

*A message queue (or simply a queue) is a mC/OS-II object that allows a task or an ISR to send pointer size variables to another task. Each pointer would typically be initialized to point to some application specific data structure containing a ‘ message’.*

**aufgabe4.c**

```
#include <hidef.h>      // for EnableInterrupts macro
#include "platform.h"   // include peripheral declarations
#if AUFGABE == 4

#define PRIO_INIT           0           // max. priority for the init task.
#define PRIO_ADC            1           // high priority for the adc task.
#define PRIO_PRODUCER       2
#define PRIO_CONSUMER       3
#define PRIO_QUEUE_STATUS   4

#define TASK_STK_SIZE       128         // Stacksize for one Task
#define MSG_QUEUE_SIZE      4           // Size of the Message Queue

OS_STK InitTaskStack[TASK_STK_SIZE];     // allocate memory for the stack of the init task
OS_STK ConsumerStack[TASK_STK_SIZE];     // allocate memory for the stack of the consumer task
OS_STK ProducerStack[TASK_STK_SIZE];     // allocate memory for the stack of the producer task
OS_STK AdcStack[TASK_STK_SIZE];          // allocate memory for the stack of the ADC task
OS_STK ShowMailStoreStack[TASK_STK_SIZE];// allocate memory for the stack of the ShowMailQueue task

OS_EVENT *msgQueue;                     // Message Queue
void *daten[MSG_QUEUE_SIZE];            // memory for the message queue

unsigned char ucDelayProducer = 10;
unsigned char ucDelayConsumer = 10;

/**
 * This task reads alternately the value of ATDCH 6 and 7 and
 * stores the 8 bit value in the variables ucDelayProducer and
 * ucDelayConsumer.
 *
```

```
 * @param pdata optional data (unused).
 */
void AdcTask(void *pdata)
{
    (void)pdata;
    for(;;)
    {
        OSTimeDly(5);
        if(ATD1SC_ATDCH == 7)
        {
            ucDelayConsumer = (ATD1RH / 4) + 1;
            ATD1SC_ATDCH=6;
        }
        else
        {
            ucDelayProducer = (ATD1RH / 4) + 1;
            ATD1SC_ATDCH=7;
        }
    }
}

/**
 * Consumer (bit D0..D3)
 *
 * @param pdata optional data (unused).
 */
void ConsumerTask(void *pdata)
{
    unsigned char ucResult;
    unsigned char ucMessage;
    (void)pdata;                        // prevents compiler warning

    for (;;)
    {
        ucMessage = (unsigned char)OSQPend(msgQueue, 0, &ucResult);

        OS_ENTER_CRITICAL();
        PTDD = (PTDD & 0xF0) | ucMessage;
        OS_EXIT_CRITICAL();

        (void)OSTaskResume(PRIO_PRODUCER);
        OSTimeDly(ucDelayConsumer);
    }
}


/**
 * Producer (bit D4..D7)
 *
 *  @param pdata optional data (unused).
 */
void ProducerTask(void *pdata)
{
    unsigned char ucBitPos = 1;
    unsigned char ucGoLeft = 1;
    unsigned char ucResult;
    (void)pdata;                        // prevents compiler warning

    for (;;)
    {
        switch (ucBitPos)
        {
            case 1: PTDD_PTDD4 = 1; break;
            case 2: PTDD_PTDD5 = 1; break;
            case 4: PTDD_PTDD6 = 1; break;
            case 8: PTDD_PTDD7 = 1; break;
        }

        if (ucGoLeft)
        {
            ucBitPos <<= 1;
            if (ucBitPos == 8) ucGoLeft = 0;
        }
        else
        {
            ucBitPos >>= 1;
            if (ucBitPos == 1) ucGoLeft = 1;
        }
```

```
        switch (ucBitPos)
        {
            case 1: PTDD_PTDD4 = 0; break;
            case 2: PTDD_PTDD5 = 0; break;
            case 4: PTDD_PTDD6 = 0; break;
            case 8: PTDD_PTDD7 = 0; break;
        }

        do
        {
            // save the message into the queue
            ucResult = OSQPost(msgQueue, (void *)(PTDD >> 4) );

            // Suspend the Task, if there is no space in the queue
            if (ucResult == OS_Q_FULL) OSTaskSuspend(OS_PRIO_SELF);
        }
        while (ucResult != OS_NO_ERR); // repeat until there is no error.

        OSTimeDly(ucDelayProducer);
    }
}


/**
 * Shows the level (0..4) at port F as a bar graph
 *
 * @param pdata optional data (unused).
 */
void ShowMsgQueue(void *pdata)
{
    INT8U count;
    OS_Q_DATA tempData;
    (void)pdata;                        // prevents compiler warning

    PTFDD = 0x0F;

    for(;;)
    {
        (void)OSQQuery(msgQueue, &tempData);
        count = (INT8U)tempData.OSNMsgs;
        PTFD = (0x0f << count);
        OSTimeDly(4);
    }
}


/**
 * Timer etc. initialisieren...
 *
 * @param pdata Optionale Daten für den Task beim Start.
 */
void InitTask(void* pdata)
{
    (void)pdata;
    OSTimerInit();                  // start timer (RTI)
    PTDD = 0xFF;                    // disable LED's on Port D
    PTDDD = 0xFF;                   // configure Port D as output

    ATD1C ATDPU=1;                  // ATD power on
    ATD1C_PRS = 2;                  // set the prescaler. Valid for a busclock of 3-12 MHz.
    ATD1C_RES8=1;                   // 8BitData
    ATD1PE_ATDPE7=1;                // Pin enabled Channel 7
    ATD1PE ATDPE6=1;                // Pin enabled Channel 6
    ATD1SC_ATDCO=1;                 // continuous conversion
    ATD1SC_ATDCH=7;                 // Select Input Channel 7

    msgQueue = OSQCreate(daten, MSG_QUEUE_SIZE);

    (void)OSTaskCreate(AdcTask, (void*)0, &AdcStack[TASK_STK_SIZE-1], PRIO_ADC);
    (void)OSTaskCreate(ConsumerTask, (void*)0, &ConsumerStack[TASK_STK_SIZE-1], PRIO_CONSUMER);
    (void)OSTaskCreate(ProducerTask, (void*)0, &ProducerStack[TASK_STK_SIZE-1], PRIO_PRODUCER);
    (void)OSTaskCreate(ShowMsgQueue, (void*)0, &ShowMailStoreStack[TASK_STK_SIZE-1],
PRIO_QUEUE_STATUS);

    (void)OSTaskDel(OS_PRIO_SELF);  // destroy the init task.
}
```

```
/**
 * main program
 *
 * This function has the following tasks:
 * - init system clock
 * - init operating system
 * - create the first (init-) task
 * - start the os
 */
void main(void)
{
    OSInit();                       // initialize the operating system

    // create the init task
    (void)OSTaskCreate(InitTask, (void*)0, &InitTaskStack[TASK_STK_SIZE-1], PRIO_INIT);

    OSStart();                      // start the operating system
}
#endif
```

# 12: Farbsensor am IIC-Bussystem

Sie vertiefen Ihre Kenntnisse zum IIC-Busprotokoll und können das IIC-Controller Modul des MC9S08JM60 zur Kommunikation mit anderen Busteilnehmern einsetzen, insbesondere dem Farbsensor.

### 1. HSV-Farbraum

Der Farbsensor des MC-Cars liefert die Farbinformation als RGB-Werte. Um eine bestimmte Farbe zu detektieren, müsste somit für drei Farbwerte (Rot, Grün, Blau) überprüft werden, ob sie in einen bestimmten Toleranzbereich fallen. Diese Toleranzbereiche sind sowohl abhängig von der Streuung der Sensoren als auch von Umgebungslichteinflüssen. Deshalb ist die Detektieren von Farben in anderen Farbräumen oft einfacher. In dieser Übung wird das HSV-Modell1 verwendet, dass sich als Zylinder repräsentieren lässt, siehe Abbildung 1.

Die 3 verwendeten Parameter besitzen dabei folgende Bedeutung:

- H = Farbton (hue) 0 … 360° 0° = Rot, 120° = Grün, 240° = Blau, Abbildung 2
- S = Farbsättigung (saturation) 0 … 100% 0% = Neutralgrau, 100% reine Farbe
- V = Hellwert (value) 0 … 100% 0% = keine Helligkeit, 100% = höchste Helligkeit

Die Zylinderachse entspricht der zentralen Graulinie mit 0% Farbsättigung und hat per Definition den Farbton H = 0 (Rot). Punkte am Aussenrand des Zylinders haben 100% Farbsättigung, wobei die Helligkeit zunimmt, je weiter oben sich der Punkt im Zylinder befindet.

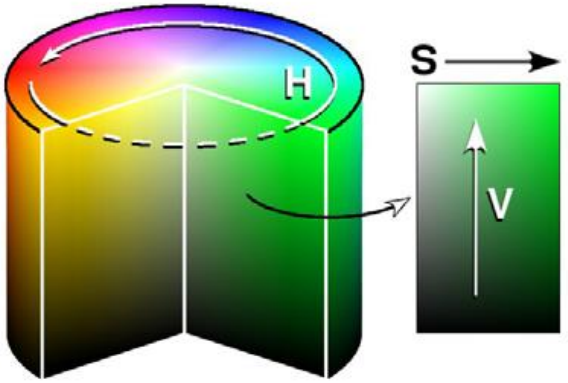



**Abbildung 1 – HSV-Farbraum**

Quelle: http://de.wikipedia.org/wiki/HSV-Farbraum


**Abbildung 2 – Farbton als Winkel 0° … 360°**

Der Vorteil des HSV-Modells kommt zum Tragen, wenn man eine bestimmte Farbe unabhängig von deren Helligkeit und Sättigung detektieren möchte. Um beispielsweise Grün zu erkennen, braucht man nur einen einzelnen Wert (H) auf Werte zwischen 100 und 140 zu prüfen. Dieser Vorteil rechtfertigt den Rechenaufwand für die Konvertierung von RGB zu HSV. Diese Umrechnung ist in der Funktion colSensRGBtoHSV() bereits fertig implementiert.

### 2. Kalibrierung

Um optimale Werte zu erreichen, sollten der Farbsensor und der Liniensensor kalibriert werden.

Die Kalibrierung wird nicht gespeichert, muss also nach jedem Reset/Einschalten durchgeführt werden. Falls das Umgebungslicht stark ändert (z.B. durch Sonneneinstrahlung), kann es auch zwischendurch notwendig sein, die Kalibrierung erneut auszuführen. Für die Kalibrierung geht man wie folgt vor:

1. MC-Car einschalten und Programm laden (falls nicht im Flash)
2. MC-Car auf eine schwarze Fläche stellen und anschliessend den Joystick-Taster kurz nach links betätigen. Hinweis: Unbedingt zuerst Schwarz kalibrieren und erst danach Weiss (wichtig bei der Berechnung der Threshhold-Werte für die Linienerkennung)!
3. MC-Car auf eine weisse Fläche stellen und anschliessend den Joystick-Taster kurz nach rechts betätigen.
4. Kalibrierung fertig, Farben werden zuverlässig erkannt

### 3. IIC-Farbsensor

Legen Sie in CW ein neues C-Projekt mit Copy/Paste an und nutzen Sie das gegebene File main.c als Hauptdatei. Fügen Sie dem neuen Projekt unter Project_Sources ausserdem die Datei colSens_temp.c zu.

Aktualisieren Sie die Bibliothek MC_Library mit den gegebenen linesens.h/c Dateien.

a) Informieren Sie sich im Datenblatt TCS3414CS.pdf über die prinzipielle Funktionsweise des Farbsensors, seine IIC-Schnittstelle und seine Register (insbesondere Command, Control und ADC Data Register).

b) Analysieren Sie den gegebenen Code und vervollständigen Sie im File colSens_temp.c die Funktionen
   1. colSensInit()
   2. colSensUpdateGain()
   3. colSensReadRawColor()

   unter Verwendung der IIC-Funktionen i2cWriteCmdData() und i2cReadCmdData() aus i2c.c (siehe CW Task-View).

c) Testen Sie Ihre Implementierung in dem Sie den MC-Car über Testfelder.png auf dem Bildschirm ihres Notebooks bewegen und dabei die wechselnden Farben der Front-LEDs beobachten.

d) Implementieren Sie in main.c andere farbabhängige Aktionen (z.B. Soundausgabe bei Rot, Beschleunigung bei Grün) für die Fahrt auf einem Parkour folgender Art:

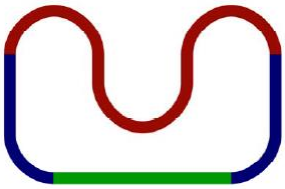


**Abbildung 3 - Parkour**

## colSens.c

```
[…]
#define I2C_COLSENS_ADR                 0x72
[…]
/**
 * Power on and ADC enable through write of appropriate bits in control register
 * Structure of communication protocol:
 * Start Condition | Byte 1 | Akn | Byte 2 | Akn | Byte 3 | Akn | Stop condition
 * contents of the transfered data
 * Byte 1           | Byte 2        | Byte 3
 * I2C Address      | Command       | Data
 * Device Adr+W     | commandReg    | Values for ControlStatusReg
 * 0x72             | 0x80          | 0x03
 *
 * @returns
 *    EC_SUCCESS or an error code if the device has not responded
 */
tError colSensInit(void)
{
  tError error;
  tCommandReg commandReg;
  tControlStatusReg controlStatusReg;

  // set default value
  whiteValue.red = whiteValue.green = whiteValue.blue = 32000;
  blackValue.red = blackValue.green = blackValue.blue = 0;

  // enable "under floor lightning"
  PTCDD_PTCDD2 = 1;
  PTCD_PTCD2 = 1;

  // @ToDo 1: Power on and enable ADC for color sensor TCS3414
  // Power on and enable ADC for color sensor TCS3414
  commandReg.Byte = 0;              // Suppress compiler warning C5651;
  commandReg.Bits.CMD = 1;          // Select command register. Must write as 1.
  commandReg.Bits.Transaction = REG_CMD_TRANS_BYTE_PROT; // use of Byte protocol for I2C
  commandReg.Bits.Address = REG_ADR_CONTROL;             // Select register for following R/W

  controlStatusReg.Byte = 0;        // Suppress compiler warning C5651;
  controlStatusReg.Bits.Power = 1;  // power on
  controlStatusReg.Bits.AdcEn = 1;  // enable ADC

  error = i2cWriteCmdData(I2C_COLSENS_ADR, commandReg.Byte, &controlStatusReg.Byte, 1);
  if (error) return error;
```

```
  // configure Gain Register in color sensor TCS3414 to the most sensitive setting
  gainReg.Bits.Gain = 3;
  gainReg.Bits.Prescaler = 0;
  return colSensUpdateGain();
}
```

**[…]**

```
/**
 * Sets Prescaler Mode and Gain in the Gain Register of TCS3414
 * (Transaction Type = Byte protocol)
 *
 * @returns
 *    EC_SUCCESS or an error code if the device has not responded
 */
tError colSensUpdateGain(void)
{
  tCommandReg commandReg;

  // @ToDo 2: Configure Gain Register
  commandReg.Byte = 0;       // Suppress compiler warning C5651;
  commandReg.Bits.CMD = 1;   // Select command register.  Must write as 1.
  commandReg.Bits.Transaction = REG_CMD_TRANS_BYTE_PROT; // use of Byte protocol for I2C
  commandReg.Bits.Address = REG_ADR_GAIN;                // select register for following read

  return i2cWriteCmdData(I2C_COLSENS_ADR, commandReg.Byte, &gainReg.Byte, sizeof(tControlGainReg));
}
```

**[…]**

```
/**
 * Reads the raw 16-bit RGB color values from the sensor TCS3414
 * in a single IIC-Read transfer.
 * (Transaction Type = Block protocol)
 *
 * @param [out] pColor
 *   pColor->red
 *   pColor->green
 *   pColor->blue
 *   pColor->clear
 *
 * @return
 *   EC_SUCCESS if it was successful, an error code otherwise
 */
tError colSensReadRawColor(tColorRawRGB *pColor)
{
  tError error;
  tCommandReg commandReg;

  // @ToDo 3: Read ADC Data Registers
  commandReg.Byte = 0;       // Suppress compiler warning C5651;
  commandReg.Bits.CMD = 1;   // Select command register.  Must write as 1.
  commandReg.Bits.Transaction = REG_CMD_TRANS_BLOCK_PROT; // use of Block protocol for I2C
  commandReg.Bits.Address = REG_ADR_DATA_GREEN;           // select register for following read

  error = i2cReadCmdData(I2C_COLSENS_ADR, commandReg.Byte, (char*)pColor, sizeof(tColorRawRGB));
  if (error) return error;

  // Note the byte ordering in the ADC Channel Data Registers!!
  swapByteOrder(&pColor->red);
  swapByteOrder(&pColor->green);
  swapByteOrder(&pColor->blue);

  return EC_SUCCESS;
}
```

**[…]**