Microcontroller
Selbststudium Semesterwoche 9

Kurzfragen zum Thema Interrupts

1. Wozu werden Interrupts benötigt (Beispiele)?
   - ➔ Interrupts werden überall dort verwendet, wo auf zeitkritische Ereignisse reagiert werden muss. Beispielsweise beim Empfangen von Daten über die serielle Schnittstelle oder über eine Netzwerkschnittstelle.
   - ➔ Wird hier nicht rechtzeitig reagiert, können Daten verloren gehen.
2. Was ist eine ISR?
   - ➔ Die Interrupt Service Routine ist ein Interrupthandler. Dieser wird von der CPU ausgeführt, wenn sie durch einen Interrupt Request (Unterbrechungsanforderung) gezwungen wird, den normalen Programmablauf zu unterbrechen und einen Interrupt auszuführen.
   - ➔ In der ISR wird Code ausgeführt, welcher beim Eintreten eines Interrupts ausgeführt werden muss.
   - ➔ Es ist zu beachten, dass innerhalb einer ISR nicht viel Code ausgeführt wird, da ansonsten andere Interrupts verzögert werden.
3. Welche Arten von Interrupts gibt es beim HCS08?
   - ➔ Reset Interrupt (Watchdog)
   - ➔ Software Interrupt
   - ➔ IRQ Pin (Hardware Interrupt)
   - ➔ Timer Interrupts
   - ➔ Serial Communication Interface Interrupts
   - ➔ Keyboard Interrupt
   - ➔ Inter Integrated Circuit Interrupts
   - ➔ Real Time Interrupt
4. Erklären Sie den Unterschied zwischen Polling und Interrupt.
   - ➔ Beim Polling Approach wird in einer Schlaufe das Eintreten eines Ereignisses geprüft. Dafür wird kostbare CPU Zeit verbraten.
   - ➔ Interrupts lösen genau auf ein bestimmtes Ereignis aus.
5. Was versteht man unter einem Interruptvektor?
   - ➔ Ein Interruptvektor enthält die Speicheradresse, an welcher – beim Eintreten eines Interrupts – gesprungen wird, um die ISR abzuarbeiten.
   - ➔ Siehe .prm Datei.
6. Worin besteht der Unterschied zwischen einer Subroutine und einer ISR?
   - ➔ Eine Subroutine kann jederzeit von jeder Stelle des Codes aufgerufen werden. Der Programmierer bestimmt, wann eine Subroutine aufgerufen wird.
   - ➔ Eine ISR wird dann aufgerufen, wenn ein entsprechender Interrupt ausgelöst wird.
7. Wieso braucht man ein Timersytem? Würde es nicht genügen, Zeitverzögerungen mit NOP-Befehlen zu programmieren?
   - ➔ Das Ausführen von NOP Befehlen würde zu einer sehr unpräzisen Zeitmessung führen und wäre für viele Aufgaben unbrauchbar.
   - ➔ Mit Timer Systemen können präzise Signale generiert werden. Beispiel: PWM. Das wäre mit NOP nicht möglich.

Aufgabe 1

Die Übungsvorlage dient zur Realisation eines Lauflichtes, das zwei wählbare Frequenzen hat, dessen Richtung wählbar ist und einen Mode beinhaltet, in dem beim Erreichen einer Randstelle die Richtung wechselt. Spezifikation:

- Anzeige Lauflicht Port C
- Frequenzwahl an Port A:0 (1: High, 0: Low)
- Port A:2,1 = 00 bedeutet eingefroren
- Port A:2,1 = 01 bedeutet "nach rechts"
- Port A:2,1 = 10 bedeutet "nach links"
- Port A:2,1 = 11 bedeutet "umkehren, wenn Randstelle erreicht", ohne Dunkelanzeige!
- Wartezeit vorerst mit einer Funktion "Delay" (Rechenzeit blockieren) realisieren

Bringen Sie das Lauflicht ab CodeWarrior-Template "Übung_9_1&2_RT" zum Funktionieren.

```
#include <hidef.h>      // for EnableInterrupts macro
#include "derivative.h" // include peripheral declarations

// Shift Modi
typedef enum {Freeze, ShiftRight, ShiftLeft, KnightRider} TShiftMode;
TShiftMode ShiftMode = Freeze;

// Frequenz Modi
typedef enum {Low, High} TFreq;
TFreq Freq = Low;

// Laufrichtung
typedef enum{left, right} TDirection;
TDirection  Laufrichtung = left;

void initClock(void)
{
    SOPT = 0x73;            // Watchdog=disabled, Stop-Mode=enabled

    ICGC1 = 0x28;           // ICG-Mode: FEI
    ICGC2 = 0x72;           // N=10, R04
    while(!ICGS1_LOCK);     // Warten bis Clock stabil (locked) ist.
}

void Init(unsigned char _Data)
{
    // Ports
    PTADD = 0x00;       // Port A: Input
    PTCDD = 0xFF;       // Port C: Output
    PTFDD = 0xFF;       // Port F: Output
    PTCD = _Data;       // Port C: Ausgabe von _Data
}
```

```
void GetControl (TShiftMode *_ShiftMode, TFreq *_Freq)
{
    // Bestimmung Frequenz Modus
    if ( (PTAD & 0x01) == 1 )
    {
        *_Freq = High;
    }
    else
    {
        *_Freq = Low;
    }
    // Bestimmung Shift Modus
    if ( (PTAD & 0x06) == 2 ) //hexadezimaler Port-Wert
    {
        *_ShiftMode = ShiftRight;
    }
    else
    {
        if ( (PTAD & 0x06) == 4 ) //hexadezimaler Port-Wert
        {
            *_ShiftMode = ShiftLeft;
        }
        else
        {
            if ( (PTAD & 0x06) == 6 ) //hexadezimaler Port-Wert
            {
                *_ShiftMode = KnightRider;
            }
            else
            {
                *_ShiftMode = Freeze;
            }
        }
    }
}

void ShowData (unsigned char *_Data, TShiftMode _ShiftMode)
{
    if ( _ShiftMode == ShiftRight )
    {
        *_Data = *_Data >> 1;
        if ( *_Data == 0 )
        {
             *_Data = 0x80; // wenn Anzeige dunkel (also Null) ist, wieder links beginnen
        }
        Laufrichtung = right;
    }
    else
    {
        if ( _ShiftMode == ShiftLeft )
        {
            *_Data = *_Data << 1;
            if ( *_Data == 0 )
            {
                *_Data = 0x01;  // wenn Anzeige dunkel ist, wieder rechts beginnen
            }
            Laufrichtung = left;
        }
        else
        {
            if ( _ShiftMode == KnightRider )
            {
                if ( Laufrichtung )
                // aktuelle Laufrichtung: rechts
                {
                    if ( *_Data == 0x01 )
                    {
                          // am rechten Display Anschlag angelangt
                        *_Data = *_Data << 1;
```

```
                        Laufrichtung = left;
                    }
                    else
                    {
                        *_Data = *_Data >> 1;
                    }
                }
                else
                // aktuelle Laufrichtung: links
                {
                    if ( *_Data == 0x80 )
                    {
                      // am linken Display Anschlag angelangt
                        *_Data = *_Data >> 1;
                        Laufrichtung = right;
                    }
                    else
                    {
                        *_Data = *_Data << 1;
                    }
                }
            }
        }
    }
    PTCD = *_Data;
}

/**
 * Wartet etwa 2 ms lang.
 */
void Delay4Hz (void)
{
    unsigned int i;
    for ( i=0;i<20000;i++ ){};
}

/**
 * Hauptprogramm
 */
void main(void)
{
    unsigned char ucData = 1;    // Ausgabe-Daten
    unsigned int i;

    initClock();                // Clock und Watchdog konfigurieren
    EnableInterrupts;           // Interrupts aktivieren

    Init(ucData);                   // Ports und A/D-Wandler konfigurieren

    for ( ;; )
    {
        GetControl (&ShiftMode, &Freq);     // Controll-Schalter einlesen
        ShowData (&ucData, ShiftMode);      // Daten berechnen und anzeigen

        // Aufgabe 1: Delay für Low/High Frequenz
        if(Freq==High) {
             Delay4Hz();
        } else {
             for(i=1;i<=4;i++) {
                   Delay4Hz();
             }
        }
    }
}
```

Aufgabe 2

Erweitern Sie das Lauflicht unter Aufgabe 1 mittels A/D Wandlersystem so, dass die Schiebefrequenz des Lauflichts mittels Potentiometer an Port B7 vorgegeben und über das A/D System eingelesen werden kann.

```
#include <hidef.h>      // for EnableInterrupts macro
#include "derivative.h" // include peripheral declarations

// Shift Modi
typedef enum {Freeze, ShiftRight, ShiftLeft, KnightRider} TShiftMode;
TShiftMode ShiftMode = Freeze;

// Frequenz Modi
typedef enum {Low, High} TFreq;
TFreq Freq = Low;

// Laufrichtung
typedef enum{left, right} TDirection;
TDirection  Laufrichtung = left;

void initClock(void)
{
    SOPT = 0x73;            // Watchdog=disabled, Stop-Mode=enabled

    ICGC1 = 0x28;           // ICG-Mode: FEI
    ICGC2 = 0x72;           // N=10, R04
    while(!ICGS1_LOCK);     // Warten bis Clock stabil (locked) ist.
}

void Init(unsigned char _Data)
{
    // Configure Ports
    PTADD = 0x00;       // Port A: Input
    PTCDD = 0xFF;       // Port C: Output
    PTFDD = 0xFF;       // Port F: Output
    PTCD = _Data;       // Port C: Ausgabe von _Data

    // Configure A/D Convertor
    ATD1C_ATDPU = 1;      // power on
    ATD1C_RES8 = 1;       // 8-bit Resolution
    ATD1C_PRS = 0x0010;   // Max BusClk = 8 MHz, Min BusClk = 3 MHz
    ATD1SC_ATDCO = 1;     // continuous conversion mode
    ATD1PE_ATDPE7 = 1;    // enable bit7 an Port
    ATD1SC_ATDCH = 0x07;  // select channel AD7
}

void GetControl (TShiftMode *_ShiftMode, TFreq *_Freq)
{
    // Bestimmung Frequenz Modus
    if ( (PTAD & 0x01) == 1 )
    {
        *_Freq = High;
    }
    else
    {
        *_Freq = Low;
    }
    // Bestimmung Shift Modus
    if ( (PTAD & 0x06) == 2 ) //hexadezimaler Port-Wert
    {
        *_ShiftMode = ShiftRight;
    }
    else
    {
        if ( (PTAD & 0x06) == 4 ) //hexadezimaler Port-Wert
        {
            *_ShiftMode = ShiftLeft;
        }
        else
```

```
        {
            if ( (PTAD & 0x06) == 6 ) //hexadezimaler Port-Wert
            {
                *_ShiftMode = KnightRider;
            }
            else
            {
                *_ShiftMode = Freeze;
            }
        }
    }
}

void ShowData (unsigned char *_Data, TShiftMode _ShiftMode)
{
    if ( _ShiftMode == ShiftRight )
    {
        *_Data = *_Data >> 1;
        if ( *_Data == 0 )
        {
             *_Data = 0x80; // wenn Anzeige dunkel (also Null) ist, wieder links beginnen
        }
        Laufrichtung = right;
    }
    else
    {
        if ( _ShiftMode == ShiftLeft )
        {
            *_Data = *_Data << 1;
            if ( *_Data == 0 )
            {
                *_Data = 0x01;  // wenn Anzeige dunkel ist, wieder rechts beginnen
            }
            Laufrichtung = left;
        }
        else
        {
            if ( _ShiftMode == KnightRider )
            {
                if ( Laufrichtung )
                // aktuelle Laufrichtung: rechts
                {
                    if ( *_Data == 0x01 )
                    {
                          // am rechten Display Anschlag angelangt
                        *_Data = *_Data << 1;
                        Laufrichtung = left;
                    }
                    else
                    {
                        *_Data = *_Data >> 1;
                    }
                }
                else
                // aktuelle Laufrichtung: links
                {
                    if ( *_Data == 0x80 )
                    {
                          // am linken Display Anschlag angelangt
                        *_Data = *_Data >> 1;
                        Laufrichtung = right;
                    }
                    else
                    {
                        *_Data = *_Data << 1;
                    }
                }
            }
        }
```

```
    }
    PTCD = *_Data;
}

/**
 * Wartet etwa 2 ms lang.
 */
void Delay2ms (void)
{
    unsigned int i;
    for ( i=0;i<200;i++ ){};
}

/**
 * Hauptprogramm
 */
void main(void)
{
    unsigned char ucData = 1;    // Ausgabe-Daten
    unsigned char ucAD_Wert;
    unsigned int uiA;

    initClock();                // Clock und Watchdog konfigurieren
    EnableInterrupts;           // Interrupts aktivieren

    Init(ucData);                   // Ports und A/D-Wandler konfigurieren

    for ( ;; )
    {
        GetControl (&ShiftMode, &Freq);     // Controll-Schalter einlesen
        ShowData (&ucData, ShiftMode);      // Daten berechnen und anzeigen

        PTFD = ATD1RH;     // AD-Wandlerwert zur Pruefung auf Port F ausgeben
        ucAD_Wert = ~PTFD;                  // ADWert ist umgekehrt prop. zu Leuchtzeit

        for (uiA=0; uiA<=ucAD_Wert; uiA++)   // AD-Wert abhängige Warteschlaufe
        {    // delay wird genau 0x aufgerufen, wenn der AD-Wert (an Poti) = 128 ist
             // (wegenInvertierung ucAD_Wert = ~PTFD)
             // delay wird genau 128x aufgerufen, wenn der AD-Wert = 0 ist.
            Delay2ms();
        }
    }
}
```