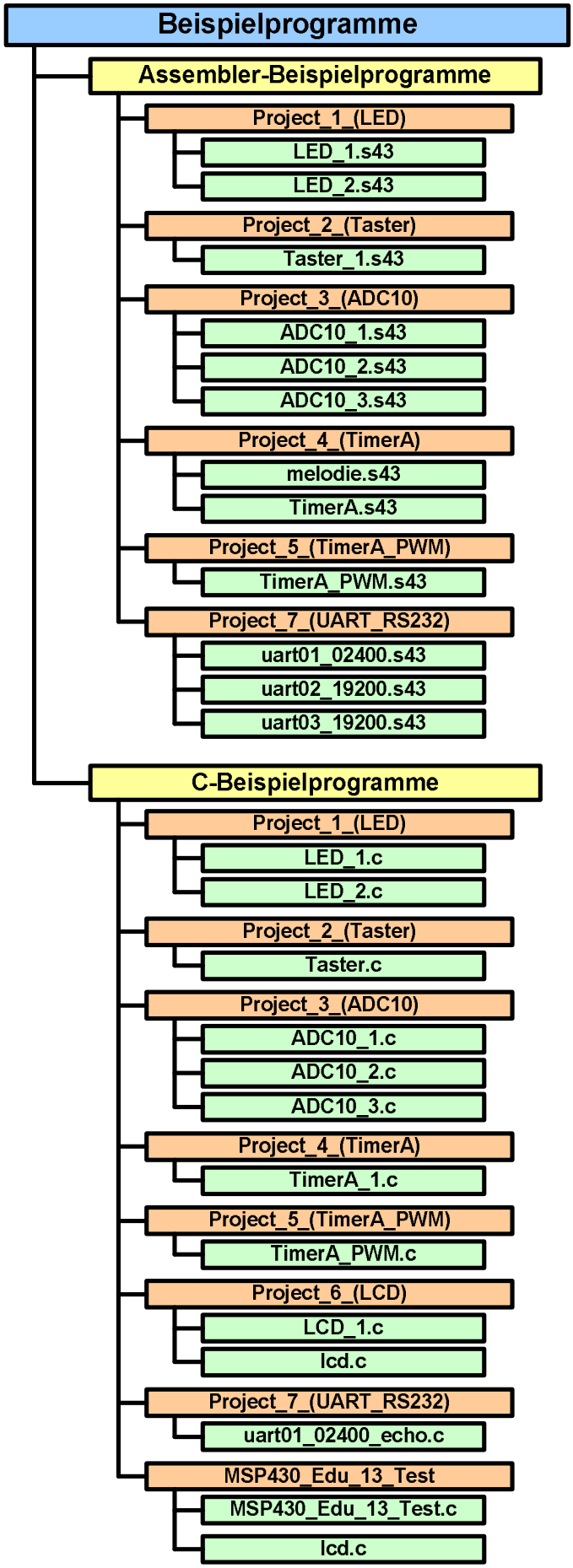
Übersicht und Erläuterung aller Beispielprogramme zum MSP430 Education System

Tobias Wengemuth

I. Inhaltsverzeichnis

1. Beispielprogramme und ihre Quelltexte

In diesem Kapitel werden die Funktionen der Test- und Simulationsprogramme kurz beschrieben und der komplett dokumentierte Quelltext aller Programme dargestellt. Durch die Kommentare gelingt es schnell, die Funktionsweise jedes einzelnen Programms zu verstehen und nachvollziehen zu können.
Alle Programme wurden jeweils für spezielle Funktionseinheiten geschrieben und können als funktionsfähiges Beispiel zu den jeweiligen Funktionseinheiten betrachtet werden. Diese Programme wurden zu Demonstrationszwecken geschrieben, um einen Einstieg in das MSP430 Education System zu finden. Diese Programme stellen keine fertigen Applikationen dar, sondern sollen eigene Entwicklungen beschleunigen. Es kommt vor, dass in einem Projekt mehrere Beispielprogramme vorhanden sind, um die Vielfältigkeit der Lösungsansätze und Applikationen zu demonstrieren. Diese können dann einzeln und nacheinander zum Testen in das Projekt einfügt und getestet werden.
Weitere Beispielprogramme können in dem Ordner FET_examples der eingesetzten IAR Embedded Workbench Programmierumgebung oder von der Texas Instruments Homepage bezogen werden. Diese Programme demonstrieren wesentlich mehr Möglichkeiten des MSP430F1232, müssen aber an das MSP430 Education System angepasst werden. Dies sollte jedoch kein Problem darstellen, da alle Funktionen und Anschlussbelegungen in dem zugehörigen Handbuch genau aufgeschlüsselt wurden.
Um alle Befehle und ihre zugehörigen Registereinstellungen richtig verstehen zu können, ist die Verwendung des MSP430 x1xx User Guides unumgänglich. In diesem werden alle Befehle mit den zu gehörigen Registern sehr genau beschrieben und die Verwendung jedes einzelnen an einem Beispiel erläutert.
Bei weiteren Fragen wird auf die beiliegenden Handbücher verwiesen.
Nachfolgend werden nun alle Assembler- und C-Beispielprogramme kurz beschrieben und dokumentiert. Alle Softwarebeschreibungen beziehen sich auf das MSP430 Education System und die IAR Embedded Workbench. Beim Einsatz anderer Compiler kann keine Kompatibilität zu den hier getroffenen Aussagen garantiert werden.



Das nebenstehende Baumdiagramm zeigt alle Beispielprogramme in einer übersichtlichen Darstellung. Dies ermöglicht ein strukturiertes und übersichtliches Arbeiten mit diesem Heft. Die Hauptbereiche (Assembler und C) wurden durch gelbe Hintergrundfarbe hervorgehoben. Alle Projektfiles werden durch eine rote Hintergrundfarbe gekennzeichnet. Alle grün hinterlegten Felder kennzeichnen die integrierten Programme. Somit ist es möglich, schnell das gewünschte Programm zu finden und die Verbindung mit dem zugehörigen Projekt zu erkennen.

Es wurden Beispielprogramme für die Programmiersprachen Assembler und C geschrieben, um unterschiedliche Funktionalitäten des Compilers und der eingesetzten Hardware zu demonstrieren.

1.1. Assembler-Beispielprogramme

In den folgenden Abschnitten werden einzelne Projekte beschrieben und verschiedene Beispielprogramme dokumentiert. Diese wurden komplett in Assembler geschrieben. Das Bedeutet, alle Register- und Portzugriffe werden durch Assemblerbefehle realisiert. Die hier aufgeführte Projektbezeichnung stellt gleichzeitig den Namen des zugehörigen Ordners dar. Die in Klammern stehenden Bezeichnungen sind die zugehörigen Projektnamen.

1.1.1. Project_1_(LED), Verwendung der acht Leuchtdioden D0 - D7

Die acht Leuchtdioden befinden sich an Port 1 des MSP430F1232 Mikrocontrollers. Alle Leuchtdioden werden hier als Ausgabe-Elemente verwendet. Somit müssen alle Pins dieses Ports im zugehörigen Portregister als Ausgang gesetzt werden. Da diese Leuchtdioden low-active arbeiten, muss an den Pins eine Null ausgegeben werden, damit die Leuchtdioden beginnen zu leuchten. Dieser Port ist 8 Bit breit. Hierdurch erfolgt die Ausgabe ebenfalls in 8-Bit Breite. In diesem Projekt werden an Hand zweier unterschiedlicher Programme, die unterschiedliche Funktionsweise dieser Ausgabeeinheit demonstriert.

1.1.1.1. LED_1.s43

MSP430F1232 Education System 1.3 - Toggle P1.0-P1.7 with Software

Der komplette Port1 wird als Ausgang geschaltet und innerhalb einer Software-Schleife gleichzeitig die Ausgangspegel der Portpins getoggelt (vertauscht). Dadurch fangen alle LEDs an Port 1 an zu blinken.

```
#include "msp430x12x2.h"
;-------------------------------------------------------------------------------
ORG0            E000h                   ; Program Start
;-------------------------------------------------------------------------------
RESET           mov.w   #0300h,SP               ; Initialize stackpointer
StopWDT         mov.w   #WDTPW+WDTHOLD,&WDTCTL  ; Stop WDT
SetupP1         bis.b   #0FFh,&P1DIR            ; P1.0-P1.7 output
                mov.b   #0FFh,&P1OUT            ; low-active LEDs off
Mainloop        xor.b   #0FFh,&P1OUT            ; Toggle P1.0-P1.7 simultaneously
Wait            mov.w   #050000,R15             ; Delay to R15
L1              dec.w   R15                     ; Decrement R15
                Jnz     L1                      ; Delay over?
                Jmp     Mainloop                ; Again
;-------------------------------------------------------------------------------
;           Interrupt Vectors Used MSP430F12x
;-------------------------------------------------------------------------------
                ORG     0FFFEh                  ; MSP430 RESET Vector
                DW      RESET                   ;
                END
```

1.1.1.2. LED_2.s43

MSP430F1232 Education System 1.3 - Flashlight on Port 1

LED Lauflicht an Port 1

- Im ersten Teil der Mainloop-Schleife werden alle LEDs ausgeschaltet (alle Ausgangspegel an Port1 auf High gesetzt, da LEDs low-active sind)
- Anschließend wird der Pegel an P1.0 auf Low gesetzt. Damit leuchtet die zugehörige LED auf. Anschließend wird die Subroutine Delay aufgerufen, die in Register R15 den Wert 30000 lädt und dann das Register bis Null dekrementiert. Danach wird der Ausgangspegel an P1.0 wieder auf High gesetzt (LED erlischt).
- Nun wird die oben beschriebene Routine für alle Portpins wiederholt. Damit läuft also ein Licht von "rechts nach links".
- Im zweiten Teil der Mainloop-Schleife -> Loop_1 wird zuerst das Register R14 mit FFh geladen. Der Inhalt von R14 wird dann in das Ausgabe-Register des Port1 geschrieben und danach wieder die Subroutine Delay aufgerufen. Anschließend wird der Inhalt von Register R14 arithmetisch nach rechts rotiert, d.h. von links eine NULL hinein geschoben. Solange R14 nicht komplett von links mit NULLEN aufgefüllt wurde, springt die Schleife immer an die Stelle Loop_1. Damit werden von links nach rechts alle LEDs nacheinander angeschaltet, bis zum Ende fast alle leuchten. Bevor dies jedoch geschehen kann, beginnt bereits wieder die Mainloop-Schleife von vorn.

```
#include "msp430x12x2.h"
;-------------------------------------------------------------------------------------------------------------------------
                ORG             0E000h                  ; Program Start
;-------------------------------------------------------------------------------------------------------------------------
RESET           mov.w           #300h,SP                ; Initialize stackpointer
StopWDT         mov.w           #WDTPW+WDTHOLD,&WDTCTL  ; Stop WDT
SetupP1         bis.b           #0FFh,&P1DIR            ; P1.0-P1.7 output

Mainloop        mov.b           #0FFh,&P1OUT            ; low-active LEDs off

                ; from right to left with bit set/reset commands
                bic.b           #BIT0,&P1OUT            ; reset P1.0 (LED on)
                call            #Delay                  ; call subroutine
                bis.b           #BIT0,&P1OUT            ; set P1.0 (LED off)
                bic.b           #BIT1,&P1OUT            ; reset P1.1 (LED on)
                call            #Delay                  ; call subroutine
                bis.b           #BIT1,&P1OUT            ; set P1.1 (LED off)
                bic.b           #BIT2,&P1OUT            ; reset P1.2 (LED on)
                call            #Delay                  ; call subroutine
                bis.b           #BIT2,&P1OUT            ; set P1.2 (LED off)
                bic.b           #BIT3,&P1OUT            ; reset P1.3 (LED on)
                call            #Delay                  ; call subroutine
                bis.b           #BIT3,&P1OUT            ; set P1.3 (LED off)
                bic.b           #BIT4,&P1OUT            ; reset P1.4 (LED on)
                call            #Delay                  ; call subroutine
```

```
                bis.b       #BIT4,&P1OUT            ; set P1.4 (LED off)
                bic.b       #BIT5,&P1OUT            ; reset P1.5 (LED on)
                call        #Delay                  ; call subroutine
                bis.b       #BIT5,&P1OUT            ; set P1.5 (LED off)
                bic.b       #BIT6,&P1OUT            ; reset P1.6 (LED on)
                call        #Delay                  ; call subroutine
                bis.b       #BIT6,&P1OUT            ; set P1.6 (LED off)
                bic.b       #BIT7,&P1OUT            ; reset P1.7 (LED on)
                call        #Delay                  ; call subroutine
                 bis.b      #BIT7,&P1OUT            ; set P1.7 (LED off)

                 ; and backwards by shifting register
                mov         #0FFh,R14
Loop_1          mov.b       R14,&P1OUT              ; move to P1 output register
                call        #Delay
                rra         R14                     ; rotate right arithmetically
                jnz         Loop_1          ; as long as register contains NOT ZERO
                jmp         Mainloop                ; Again

Delay
                mov.w       #030000,R15             ; Delay to register R15
Delay_1         dec.w       R15                     ; Decrement register R15
                jnz         Delay_1                 ; Delay over? ZERO
end_Delay
                ret
;-------------------------------------------------------------------------------------------------------------------------------
;               Interrupt Vectors Used MSP430F12x
;-------------------------------------------------------------------------------------------------------------------------------
                ORG         0FFFEh                  ; MSP430 RESET Vector
                DW          RESET                   ;
                END
```

1.1.2. Project_2_(Taster), Verwendung der vier Taster S2 - S5

Die Taster S2 – S5 befinden sich an Port 2 des MSP430F1232. Mit Hilfe dieses Projektes soll eine entsprechende Tasterfunktion demonstriert werden.

1.1.2.1. Taster_1.s43

MSP430F1232 Education System 1.3 - Button Demo

Alle Pins an Port 1 sind als Ausgang geschaltet. Für alle die Pins (Port 2), an denen Taster angeschlossen sind, wird die Generierung von Interrupts erlaubt. Danach geht der MSP430 in den Low-Power-Mode.
Durch Betätigung eines Tasters wird der Controller aufgeweckt und generiert einen Port2 Interrupt. Innerhalb der Interrupt-Routine wird zuerst, durch Aufruf der Subroutine DebounceDelay, eine gewisse Zeit gewartet. Dies dient der Taster-Entprellung. Anschließend wird gestestet, welcher Eingangsport (und damit verbunden, welcher Taster) den Interrupt generiert hat (bzw. welcher Taster gedrückt wurde). Entsprechend wird dann der Ausgangspegel eines Port1 Pins getoggelt und die zugehörige LED ein- bzw. ausgeschaltet.

Wichtig:
Die Sprungmarke für den Port2 Interrupt 'P2_ISR' muss in die Interrupt-Vektor-Tabelle an der richtigen Adresse eingetragen werden.

```
#include "msp430x12x2.h"
;-------------------------------------------------------------------------------------------------------------------------
                ORG             0E000h                  ; Program Start
;-------------------------------------------------------------------------------------------------------------------------
RESET           mov.w           #300h,SP                ; Initialize stackpointer
StopWDT         mov.w           #WDTPW+WDTHOLD,&WDTCTL      ; Stop WDT
SetupP1         mov.b           #0FFh,&P1DIR            ; P1.0-P1.7 output
                mov.b           #0FFh,&P1OUT            ; low-active LEDs off
SetupP2         bis.b           #BIT0+BIT1+BIT2+BIT5,&P2IE ; enable P2.0, P2.1, P2.2, P2.5
                                                        ; interrupts
                eint                                    ; enable general interrupt

Mainloop        bis.w           #CPUOFF,SR              ; CPU off
                nop                                     ; Required for C-spy




;-------------------------------------------------------------------------------------------------------------------------
; P2 Interrupt Service Routine
```

```
;-------------------------------------------------------------------------------------------------------------------------
P2_ISR
                call        #DebounceDelay          ; dummy method to debounce
                                                    ; buttons
                bit.b       #BIT0,&P2IFG            ; test if P2.0 caused the ISR
                jz          P2_ISR_1                ; jump if not
                xor.b       #BIT0,&P1OUT            ; toggle LED on P1.0
                bic.b       #BIT0,&P2IFG            ; reset P2.0 interrupt flag
                reti                                ; return from interrupt
P2_ISR_1        bit.b       #BIT1,&P2IFG            ; test if P2.1 caused the ISR
                jz          P2_ISR_2                ; jump if not
                xor.b       #BIT1,&P1OUT            ; toggle LED on P1.1
                bic.b       #BIT1,&P2IFG            ; reset P2.1 interrupt flag
                reti                                ; return from interrupt
P2_ISR_2        bit.b       #BIT2,&P2IFG            ; and so on...
                 jz         P2_ISR_3
                xor.b       #BIT2,&P1OUT
                bic.b       #BIT2,&P2IFG
                reti
P2_ISR_3        bit.b       #BIT5,&P2IFG
                jz          end_P2_ISR
                xor.b       #BIT3,&P1OUT
                bic.b       #BIT5,&P2IFG
                reti
end_P2_ISR      clr.b       &P2IFG
                reti
;-------------------------------------------------------------------------------------------------------------------------
; DebounceDelay Subroutine
;-------------------------------------------------------------------------------------------------------------------------
DebounceDelay
                mov.w       #050000,R15             ; Delay to register R15
Delay_1         dec.w       R15                     ; Decrement register R15
                jnz         Delay_1                 ; Delay over? ZERO
end_DebounceDelay
                ret
;-------------------------------------------------------------------------------------------------------------------------
; Interrupt Vectors Used MSP430F12x
;-------------------------------------------------------------------------------------------------------------------------
                ORG         0FFFEh                  ; MSP430 RESET Vector
                DW          RESET                   ;
                ORG         0FFE6h                  ; MSP430 P2 Interrupt Vector
                DW          P2_ISR
                END
```

1.1.3. Project_3_(ADC10), Verwendung der Potentiometer R37 und R39

In diesem Projekt gibt es zwei Projektfiles, das ADC10_1.prj und das ADC_2.prj. Beide Projektfiles sind identisch, laden jedoch ein jeweils anderes Programm. Es wurde sich für diese Lösung entschieden, damit beide Potentiometer mit jeweils einem eigenen Projekt geöffnet und verwendet werden können. Die Voreinstellungen sind jeweils die gleichen. Alle drei Beispielprogramme können natürlich auch von einem Projekt aus verwaltet werden.
Mit diesen Programmen wird die Funktion des internen A/D-Wandlers an Port 3 demonstriert. Die angeschlossenen Potentiometer stellen eine Spannung an dem jeweiligen Port bereit, die der A/D-Wandler dann in einen digitalen Wert umwandelt und vom Mikrocontroller weiter verarbeitet werden kann. Als Ausgabebaugruppen werden die LEDs sowie das an Port 2 angeschlossene Analoge Drehspulanzeigenelement verwendet.

1.1.3.1. ADC10_1.s43

MSP430F1232 Education System 1.3 - ADC10 Demo (1)

AD-Wandlung des Spannungspegels an A6, Potentiometer 2 (P3.6, R37)

Die Samplefrequenz wird getriggert durch den CCR0 Interrupt. Innerhalb des CCR0 Interrupt wird eine neue AD-Wandlung initialisiert und freigegeben. Das Triggern des ADC10 Sampling-Timers erfolgt durch den CCR2 Block. Sobald der AD-Wandler mit der Wandlung fertig ist, wird seine Interrupt Routine 'ADC10_ISR' ausgeführt. Darin wird mit einem ausgeklügelten Algorithmus der Wandlungswert als LED-Balken dargestellt. Der Wandlungswert wird zudem verwendet, um den Duty Cycle der CCR1 PWM zu verändern. Das CCR1 pulsmodulierte Signal ist auf das analoge Anzeigeninstrument geschaltet.

```
#include "msp430x12x2.h"
#define PWM_offset 10
;-------------------------------------------------------------------------------------------------------------------------
                ORG             0E000h                  ; Program Start
;-------------------------------------------------------------------------------------------------------------------------
RESET           mov.w           #0300h,SP               ; Initialize stackpointer
StopWDT         mov.w           #WDTPW+WDTHOLD,&WDTCTL  ; Stop WDT
SetupADC10      mov.w           #INCH_6+SHS_3,&ADC10CTL1    ; P3.6, TA2 trigger sample start
                bis.b           #BIT6,&ADC10AE          ; P3.6 ADC10 option select
SetupP1         mov.b           #0FFh,&P1DIR            ; P1.0 output
SetupP2         bis.b           #BIT3,&P2DIR
                bis.b           #BIT3,&P2SEL
SetupC0         mov.w           #CCIE,&CCTL0            ; Enable interrupt
                mov.w           #03FFh+PWM_offset,&CCR0 ; PWM Period
SetupC1         mov.w           #OUTMOD_7,&CCTL1        ; CCR1 reset/set
                mov.w           #1,&CCR1                ; CCR1 PWM Duty Cycle
SetupC2         mov.w           #OUTMOD_3,&CCTL2        ; CCR2 set/reset
```

```
                mov.w   #2,&CCR2                ; CCR2 PWM Duty Cycle
SetupTA         mov.w   #TASSEL0+MC0,TACTL      ; ACLK, up mode
                                                ;
Mainloop        bis.w   #LPM3+GIE,SR            ; Enter LPM3
                nop                             ; Required only for C-spy
;-------------------------------------------------------------------------------
ADC10_ISR;
;-------------------------------------------------------------------------------
                bic.w   #ENC,&ADC10CTL0         ; ADC10 disabled
                clr.w   &ADC10CTL0              ; ADC10, Vref disabled completely
                mov.w   #07EFh,R4               ; initialize R4
                mov.w   #01FFh,R5               ; initialize R4
                mov.w   &ADC10MEM,R6            ; move ADC value to R6
                add.w   #PWM_offset,R6          ; add offset to R6
ADC10_ISR_1
                rra     R4                      ; rotate R4 arithmetically
                rra     R5                      ; rotate R5 arithmetically
                jz      ADC10_ISR_2             ; jump if R5 = 0
                cmp.w   R4,&ADC10MEM            ; compare ADC value with R4
                jlo     ADC10_ISR_1             ; jump if lower
ADC10_ISR_2
                mov.b   R5,&P1OUT               ; move R5 to P1 output register
ADC10_Exit
                mov.w   R6,&CCR1                ; change CCR1 PWM Duty Cycle
                                                ; according to ADC value
                reti
;-------------------------------------------------------------------------------
TA0_ISR;
;-------------------------------------------------------------------------------
                mov.w   #SREF_0+ADC10SHT_2+REFON+ADC10ON+ADC10IE,
                        &ADC10CTL0;
                bis.w   #ENC,&ADC10CTL0         ; ADC10 enable set
                reti                            ;
;-------------------------------------------------------------------------------
; Interrupt Vectors Used MSP430x12x2
;-------------------------------------------------------------------------------
                ORG     0FFFEh                  ; MSP430 RESET Vector
                DW      RESET                   ;
                ORG     0FFEAh                  ; ADC10 Vector
                DW      ADC10_ISR               ;
                ORG     0FFF2h                  ; Timer_A0 Vector
                DW      TA0_ISR                 ;
                END
```

1.1.3.2. ADC10_2.s43

MSP430F1232 Education System 1.3 - ADC10 Demo (2)

AD-Wandlung des Spannungspegels an A7, Potentiometer 1 (P3.7, R39)

Die Samplefrequenz wird getriggert durch den CCR0 Interrupt. Innerhalb des CCR0 Interrupt wird dann eine neue AD-Wandlung initialisiert und freigegeben. Das Triggern des ADC10 Sampling Timers erfolgt durch den CCR2 Block. Sobald der AD-Wandler mit der Wandlung fertig ist, wird seine Interrupt Routine 'ADC10_ISR' ausgeführt. Es wird kontrolliert, ob der Wandlungswert über oder unter einem bestimmten Schwellwert liegt und entsprechend die LED an P1.0 an- oder ausgeschaltet. Anschließend wird der Wandlungswert verwendet, um den Duty Cycle der CCR1 PWM zu verändern. Das CCR1 pulsmodulierte Signal ist auf das analoge Anzeigeninstrument geschaltet.

```
#include "msp430x12x2.h"
#define PWM_offset 10
;-------------------------------------------------------------------------------------------------------------------------
                ORG             0E000h                  ; Program Start
;-------------------------------------------------------------------------------------------------------------------------
RESET           mov.w           #0300h,SP               ; Initialize stackpointer
StopWDT         mov.w           #WDTPW+WDTHOLD,&WDTCTL  ; Stop WDT
SetupADC10      mov.w           #INCH_7+SHS_3,&ADC10CTL1 ; P3.7, TA2 trigger sample start
                bis.b           #BIT7,&ADC10AE          ; P3.7 ADC10 option select
SetupP1         bis.b           #BIT0,&P1DIR            ; P1.0 output
SetupP2         bis.b           #BIT3,&P2DIR            ; P2.3 output
                bis.b           #BIT3,&P2SEL            ; select P2.3 modul function (TA1)
SetupC0         mov.w           #CCIE,&CCTL0            ; Enable interrupt
                mov.w           #03FFh,&CCR0            ; PWM Period
SetupC1         mov.w           #OUTMOD_7,&CCTL1        ; CCR1 reset/set
                mov.w           #1,&CCR1                ; CCR1 PWM Duty Cycle
SetupC2         mov.w           #OUTMOD_3,&CCTL2        ; CCR2 set/reset
                mov.w           #2,&CCR2                ; CCR2 PWM Duty Cycle
SetupTA         mov.w           #TASSEL0+MC0,TACTL      ; ACLK, start timer in up mode
                                                        ;
Mainloop        bis.w           #LPM3+GIE,SR            ; Enter LPM3
                nop                                     ; Required only for C-spy
;-------------------------------------------------------------------------------------------------------------------------
ADC10_ISR;
;-------------------------------------------------------------------------------------------------------------------------
                bic.w           #ENC,&ADC10CTL0         ; ADC10 disabled
                clr.w           &ADC10CTL0              ; ADC10, Vref disabled completely
                bic.b           #01h,&P1OUT             ; P1.0 = 0 (LED on)
                cmp.w           #01FFh,&ADC10MEM        ; ADC10MEM = A0 > 0.2V?
                jlo             ADC10_Exit              ; Again
                bis.b           #01h,&P1OUT             ; P1.0 = 1 (LED off)
ADC10_Exit
                mov.w           &ADC10MEM,R6            ; move ADC value to R6
                add.w           #PWM_offset,R6          ; add offset to R6
                mov.w           R6,&CCR1                ; change CCR1 PWM Duty Cycle
                                                        ; according to ADC value
```

```
                reti
;-------------------------------------------------------------------------------------------------------------------------
TA0_ISR;
;-------------------------------------------------------------------------------------------------------------------------
                mov.w       #SREF_0+ADC10SHT_2+REFON+ADC10ON+ADC10IE,
                            &ADC10CTL0;
                bis.w       #ENC,&ADC10CTL0         ; ADC10 enable set
                reti                                ;
;-------------------------------------------------------------------------------------------------------------------------
; Interrupt Vectors Used MSP430x12x2
;-------------------------------------------------------------------------------------------------------------------------
                ORG          0FFFEh                 ; MSP430 RESET Vector
                DW          RESET                   ;
                ORG         0FFEAh                  ; ADC10 Vector
                DW          ADC10_ISR               ;
                ORG         0FFF2h                  ; Timer_A0 Vector
                DW          TA0_ISR                 ;
                END
```

1.1.3.3. ADC10_3.s43

MSP430F1232 Education System 1.3 - ADC10 Demo (3)

AD-Wandlung des Spannungspegels an A7

Die Samplefrequenz wird getriggert durch den CCR0 Interrupt. Innerhalb des CCR0 Interrupt wird eine neue AD-Wandlung initialisiert und freigegeben. Das Triggern des ADC10 Sampling Timers erfolgt durch den CCR2 Block. Sobald der AD-Wandler mit der Wandlung fertig ist, wird seine Interrupt Routine 'ADC10_ISR' ausgeführt. Darin wird mit einem ausgeklügelten Algorithmus der Wandlungswert als LED-Balken dargestellt. Der Wandlungswert wird zudem verwendet, um den Duty Cycle der CCR1 PWM zu verändern. Das CCR1 pulsmodulierte Signal ist auf das analoge Anzeigeninstrument geschaltet.
In diesem Programm wird jedoch auf den Gebrauch des Offset (PWM_offset) verzichtet.

```
#include "msp430x12x2.h"
;-------------------------------------------------------------------------------------------------------------------------
                ORG             0E000h                  ; Program Start
;-------------------------------------------------------------------------------------------------------------------------
RESET           mov.w           #0300h,SP               ; Initialize stackpointer
StopWDT         mov.w           #WDTPW+WDTHOLD,&WDTCTL  ; Stop WDT
SetupADC10      mov.w           #INCH_7+SHS_1,&ADC10CTL1 ; P3.7, TA1 trigger sample start
                bis.b           #BIT7,&ADC10AE          ; P3.7 ADC10 option select
SetupP1         mov.b           #0FFh,&P1DIR            ; P1.0 output
SetupC0         mov.w           #CCIE,&CCTL0            ; Enable interrupt
                mov.w           #32-1,&CCR0             ; PWM Period
SetupC1         mov.w           #OUTMOD1+OUTMOD0,&CCTL1  ; CCR1 set/reset
                mov.w           #2,&CCR1                ; CCR1 PWM Duty Cycle
SetupTA         mov.w           #TASSEL0+MC0,TACTL      ; ACLK, up mode
                                                        ;
Mainloop        bis.w           #LPM3+GIE,SR            ; Enter LPM3
                nop                                     ; Required only for C-spy
;-------------------------------------------------------------------------------------------------------------------------
ADC10_ISR;
;-------------------------------------------------------------------------------------------------------------------------
                bic.w           #ENC,&ADC10CTL0         ; ADC10 disabled
                clr.w           &ADC10CTL0              ; ADC10, Vref disabled completely
                mov.w           #07EFh,R4               ; initialize R4
                mov.w           #01FFh,R5               ; initialize R4
ADC10_ISR_1
                rra             R4                      ; rotate R4 arithmetically
                rra             R5                      ; rotate R5 arithmetically
                jz              ADC10_ISR_2             ; jump if R5 = 0
                cmp.w           R4,&ADC10MEM            ; compare ADC value with R4
                jlo             ADC10_ISR_1             ; jump if lower
ADC10_ISR_2
                mov.b   R5,&P1OUT                       ; move R5 to P1 output register
end_ADC10_ISR
                reti
;-------------------------------------------------------------------------------------------------------------------------
TA0_ISR;
```

```
;-------------------------------------------------------------------------------------------------------------------
                mov.w       #SREF_0+ADC10SHT_2+REFON+ADC10ON+ADC10IE,
                            &ADC10CTL0;
                bis.w       #ENC,&ADC10CTL0             ; ADC10 enable set
                reti                ;
;-------------------------------------------------------------------------------------------------------------------
; Interrupt Vectors Used MSP430x12x2
;-------------------------------------------------------------------------------------------------------------------
                ORG         0FFFEh                      ; MSP430 RESET Vector
                DW          RESET                       ;
                ORG         0FFEAh                       ; ADC10 Vector
                DW          ADC10_ISR                   ;
                ORG         0FFF2h                      ; Timer_A0 Vector
                DW          TA0_ISR                     ;
                END
```

1.1.4. Project_4_(TimerA), Verwendung des internen Timers

Der Timer ist eine sehr vielseitig einsetzbare, interne Baugruppe des MSP430F1232. Dadurch können verschiedene Zeiten, zur späteren Verwendung in einem eigenen Programm, generiert werden. Somit ist es möglich, verschiedene Programme mit einer einheitlichen Zeitbasis zu programmieren. Timer sind ebenfalls zur Generierung von Tönen sehr wichtig, da durch ihre Verwendung Pulsweiten sehr einfach eingestellt werden können, die später im Programm weiter verarbeitet werden können.
Die nachfolgenden Programme zeigen die Verwendung dieser Baugruppe an Hand des Lautsprechers an Port 3 sowie der LEDs an Port 1 des MSP430 Education System.

1.1.4.1. melodie.s43

MSP430F1232 Education System 1.3 - Sound Demo

Das CCR2-Modul wird dazu verwendet, Töne zu generieren. Den Tasten des Education Systems sind verschiedene Töne zugeordnet. Durch Drücken einer Taste wird das Timer_A Modul gestartet und die Variablen sec_counter und tone_steps entsprechend des zu generierenden Tones und dessen Dauer initialisiert.

```
#include "msp430x12x2.h"

                RSEG UDATA0
sec_counter   ds  1
tone_steps    ds  1
;-------------------------------------------------------------------------------------------------------------------------
                RSEG CODE                                       ; Program Start
;-------------------------------------------------------------------------------------------------------------------------
RESET           mov.w       #300h,SP                            ; Initialize stackpointer
StopWDT         mov.w       #WDTPW+WDTHOLD,&WDTCTL     ; Stop WDT
SetupP2         bis.b       #BIT0+BIT1+BIT2+BIT5,&P2IE ; enable P2.0, P2.1, P2.2, P2.5
                                                                ; interrupts
                eint                                            ; Enable interrupts
Mainloop        bis.w       #CPUOFF,SR                          ; Enter LPM3
                nop                                             ; Required for C-spy

;-------------------------------------------------------------------------------------------------------------------------
SetupTA;
;-------------------------------------------------------------------------------------------------------------------------
                mov.w       #TASSEL0+TACLR+TAIE,&TACTL      ; ACLK, clear TAR, interrupt
SetupC0         mov.w       #OUTMOD_4 +CCIE,&CCTL0  ; CCR0 toggle, interrupt enabled
SetupC2         mov.w       #OUTMOD_4 +CCIE,&CCTL2  ; CCR2 toggle, interrupt enabled
                mov.w       #32768,&CCR0                        ; init CCR0 Duty Cycle
                mov.b       R12,tone_steps                      ; move R12 to tone_steps
                mov.w       R12,&CCR2                           ; move R12 to CCR2
                mov.b       R14,sec_counter                     ; move R14 to sec_counter
SetP2           bis.b       #BIT4,&P2SEL                        ; select P2.4 module function TA2
                bis.b       #BIT4,&P2DIR                        ; P2.4 as output
```

```
                bis.w       #MC1,&TACTL             ; start Timer_a in continous mode
end_SetupTA
                ret

;-------------------------------------------------------------------------------------------------------------------------------
DebounceDelay;
;-------------------------------------------------------------------------------------------------------------------------------
                mov.w       #050000,R15             ; Delay to register R15
Delay_1         dec.w       R15                     ; Decrement register R15
                jnz         Delay_1                 ; Delay over? ZERO
end_DebounceDelay
                ret
;-------------------------------------------------------------------------------------------------------------------------------
; P2 Interrupt Service Routine
;-------------------------------------------------------------------------------------------------------------------------------
P2_ISR
                call        #DebounceDelay          ; dummy method to debounce buttons
                bit.b       #BIT0,&P2IFG            ; test if P2.0 caused the ISR
                jz          P2_ISR_1                ; jump if not
                mov         #37,R12                 ; init R12 with value for tone
                mov         #1,R14                  ; init R14 with value for seconds
                call        #SetupTA                ; call subroutine
                bic.b       #BIT0,&P2IFG            ; reset P2.0 interrupt flag
                reti                                ; return from interrupt
P2_ISR_1        bit.b       #BIT1,&P2IFG            ; test if P2.1 caused the ISR
                jz          P2_ISR_2                ; jump if not
                mov         #18,R12                 ; init R12 with value for tone
                mov         #1,R14                  ; init R14 with value for seconds
                call        #SetupTA                ; call subroutine
                bic.b       #BIT1,&P2IFG            ; reset P2.1 interrupt flag
                reti
P2_ISR_2        bit.b       #BIT2,&P2IFG            ; and so on...
                jz          P2_ISR_3
                mov         #9,R12; tone
                mov         #1,R14
                call        #SetupTA
                bic.b       #BIT2,&P2IFG
                reti
P2_ISR_3        bit.b       #BIT5,&P2IFG
                jz          end_P2_ISR
                mov         #5,R12; tone
                mov         #1,R14
                call        #SetupTA
                bic.b       #BIT5,&P2IFG
                reti
end_P2_ISR      clr.b       &P2IFG                  ; reset all P2 interrupt flags
                reti
;-------------------------------------------------------------------------------------------------------------------------------
TA0_ISR;
;-------------------------------------------------------------------------------------------------------------------------------
                add.w       #32768,&CCR0            ; Offset until next interrupt
                dec.b       sec_counter             ; decrement counter
                jz          TA0_ISR_1               ; jump if counter = 0
                reti
TA0_ISR_1       clr         &TACTL                  ; stop Timer A
                reti
;-------------------------------------------------------------------------------------------------------------------------------
TAX_ISR;    Common ISR for CCR1-4 and overflow
;-------------------------------------------------------------------------------------------------------------------------------
                add.w       &TAIV,PC                ; Add Timer_A offset vector
                reti                                ; CCR0 - no source
```

```
                jmp         CCR1_ISR                ; CCR1
                jmp         CCR2_ISR                ; CCR1
                reti                                ; CCR3
                reti                                ; CCR4
TA_over         xor.b       #001h,&P1OUT            ; Toggle P1.0
                reti                                ; Return from overflow ISR
CCR1_ISR        nop
                reti                                ; Return ISR
CCR2_ISR
                mov.b       tone_steps,R5           ; move tone_steps to R5
                add.w       R5,&CCR2                ; Offset until next interrupt
                reti                                ; Return ISR
;-------------------------------------------------------------------------------------------------------------------------------
; Interrupt Vectors Used MSP430x12x(2)
;-------------------------------------------------------------------------------------------------------------------------------
        COMMON  INTVEC(1)
                ORG         RESET_VECTOR            ; MSP430 RESET Vector
                DW          RESET                   ;
                ORG         TIMERA0_VECTOR          ; Timer_A0 Vector
                DW          TA0_ISR                 ;
                ORG         TIMERA1_VECTOR          ; Timer_AX Vector
                DW          TAX_ISR                 ;
                ORG         PORT2_VECTOR            ; MSP430 P2 Interrupt Vector
                DW          P2_ISR
                END
```

1.1.4.2. TimerA.s43

MSP430F1232 Education System 1.3 - Timer_A Interrupt Demo

Toggle P1.0,P1.1,P1.2,P1.3, overflow ISR, compare CCR0, CCR1, CCR2, 32kHz ACLK

Alle zur Verfügung stehenden Timer_A Interrupt Quellen werden dazu verwendet, die LEDs blinken zu lassen. Dabei ist zu beachten, dass das CCR0 Modul eine eigene Interrupt Routine hat und sich TimerOverflow, CCR1 und CCR2 eine gemeinsame Interrupt Routine teilen.

```
#include "msp430x12x2.h"
;-------------------------------------------------------------------------------------------------------------------------
                ORG     0E000h                  ; Program Start
;-------------------------------------------------------------------------------------------------------------------------
RESET           mov.w   #300h,SP                ; Initialize stackpointer
StopWDT         mov.w   #WDTPW+WDTHOLD,&WDTCTL  ; Stop WDT
SetupTA         mov.w   #TASSEL0+TACLR+TAIE,&TACTL  ; ACLK, Clr TAR, interrupt
                mov.w   #CCIE,&CCTL0            ; enable CCR0 compare interrupt
                mov.w   #07FFFh,&CCR0           ; initialize CCR0
                mov.w   #CCIE,&CCTL1            ; enable CCR1 compare interrupt
                mov.w   #03FFFh,&CCR1           ; initialize CCR1
                mov.w   #CCIE,&CCTL2            ; enable CCR2 compare interrupt
                mov.w   #01FFFh,&CCR2           ; initialize CCR2
SetupP1         bis.b   #BIT0+BIT1+BIT2+BIT3,&P1DIR ; P1.0-P1.3 output
                mov.b   #0FFh,&P1OUT            ; P1 as output
StartTA         bis.w   #MC1,&TACTL             ; Start timer_A continous mode
                eint                            ; Enable interrupts
                                                ;
Mainloop        bis.w   #LPM3,SR                ; MCLK is not required
                nop                             ; Required only for C-spy

;-------------------------------------------------------------------------------------------------------------------------
TA_CCR0_ISR;    ISR for CCR0
;-------------------------------------------------------------------------------------------------------------------------
                add.w   #07FFFh,&CCR0           ; add CCR0 init value to CCR0
                XOR.b   #BIT1,&P1OUT            ; toggle P1.1
end_TA_CCR0_ISR
                reti
;-------------------------------------------------------------------------------------------------------------------------
TAX_ISR;    Common ISR for CCR1-4 and overflow
;-------------------------------------------------------------------------------------------------------------------------
                ADD     &TAIV,PC                ; Add offset to Jump table
                RETI                            ; Vector 0: No interrupt
                JMP     TIMMOD1                 ; Vector 2: Module 1
                JMP     TIMMOD2                 ; Vector 4: Module 2
                JMP     TIMMOD3                 ; Vector 6: Module 3
                JMP     TIMMOD4                 ; Vector 8: Module 4
                                                ; Module 5. Timer Overflow Handler
                                                ; fall through
TIMOVH                                          ; Vector 10: TIMOV Flag
                XOR.b   #BIT0,&P1OUT            ; toggle P1.0
                RETI                            ;
```

```
TIMMOD1                                                 ; Vector 2: Module 1
                add.w       #03FFFh,&CCR1               ; add CCR1 init value to CCR1
                XOR.b       #BIT2,&P1OUT                ; toggle P1.2
                RETI
TIMMOD2                                                 ; Vector 4: Module 2
                add.w       #01FFFh,&CCR2               ; add CCR2 init value to CCR2
                XOR.b       #BIT3,&P1OUT                ; toggle P1.3
TIMMOD3
TIMMOD4
                RETI                                    ;
;-------------------------------------------------------------------------------------------------------------------------------
; Interrupt Vectors Used MSP430F12x
;-------------------------------------------------------------------------------------------------------------------------------
                ORG         0FFFEh                      ; MSP430 RESET Vector
                DW          RESET                       ;
                ORG         0FFF0h                      ; Timer_AX Vector
                DW          TAX_ISR                     ;
                ORG         0FFF2h                      ; Timer_A0 Vector
                DW          TA_CCR0_ISR
                END
```

1.1.5. Project_5_(TimerA_PWM),

In diesem Projekt wird ebenfalls der Timer_A verwendet, hier jedoch zum Generieren eines pulsweitenmodulierten Signals. Durch den Einsatz des Timers gelingt es, wie schon bei dem Lautsprecher, sehr schnell eine gewünschte Pulsweite einzustellen.

1.1.5.1. TimerA_PWM.s43

MSP430F1232 Education System 1.3 - Timer_A Pulse Width Modulation

Dieses Programm zeigt die einfachste Variante einer PWM. Timer_A wird im up-mode betrieben, d.h. er zählt bis zum Wert, der im Register CCR0 (30000) gespeichert ist und beginnt dann wieder bei 0. Die Ausgabeeinheiten der Module CCR1 und CCR2 werden auf reset/set initialisiert. Die Werte im CCR1 und CCR2 Register bestimmen die Pulsweite der Ausgangssignale, wobei die Periodendauer durch den Wert im CCR0 Register und die Taktung des Timer_A (ACLK) bestimmt wird. ACLK steht für den 32 kHz Quarz und wird in diesem Programm als TACLK verwendet. Daraus ergibt sich bei einer 75% und 25% Pulsdauer PWM eine Periodendauer von 15,6 ms.

```
#include "msp430x12x2.h"
;-------------------------------------------------------------------------------------------------------------------------
                ORG             0E000h                  ; Program Start
;-------------------------------------------------------------------------------------------------------------------------
RESET           mov.w           #300h,SP                ; Initialize stackpointer
StopWDT         mov.w           #WDTPW+WDTHOLD,&WDTCTL  ; Stop WDT
SetupTA         mov.w           #TASSEL0+TACLR,&TACTL   ; ACLK, Clear TAR
SetupC0         mov.w           #OUTMOD_4,&CCTL0        ; toggle
                mov.w           #30000,&CCR0            ; PWM Period
SetupC1         mov.w           #OUTMOD_7,&CCTL1        ; CCR1 reset/set
                mov.w           #22500,&CCR1            ; CCR1 PWM Duty Cycle
SetupC2         mov.w           #OUTMOD_7,&CCTL2        ; CCR2 reset/set
                mov.w           #7500,&CCR2             ; CCR2 PWM duty cycle
SetupP1         bis.b           #BIT1+BIT2+BIT3,&P1DIR  ; P1.2 and P1.3 outputs
                bis.b           #BIT1+BIT2+BIT3,&P1SEL  ; P1.2 and P1.3 TA1/2 option
                bis.w           #MC0,&TACTL             ; Start Timer_A in upmode
                                                        ;
Mainloop        bis.w           #LPM3,SR                ; Enter LPM3
                nop                                     ; Required only for C-spy
                                                        ;
;-------------------------------------------------------------------------------------------------------------------------
; Interrupt Vectors Used MSP430x12x(2)
;-------------------------------------------------------------------------------------------------------------------------
                ORG             0FFFEh                  ; MSP430 RESET Vector
                DW              RESET                   ;
                END
```

1.1.6. Project_7_(UART_RS232), Funktion der UART-Schnittstelle

Die Serielle Kommunikation wird über eine RS-232 Schnittstelle realisiert. Diese Schnittstelle wird über das UART-Modul des Mikrocontrollers angesprochen und verwendet. Durch diese Schnittstelle können mehrere MSP430 Education Systeme miteinander verbunden, Daten mit einem Computer oder einen anderen Gerät mit einer RS-232 Schnittstelle ausgetauscht und externe Baugruppen ansteuert werden.
Das UART Modul wird in Zusammenhang mit dem Hyperterminal des Computers verwendet. Im Hyperterminal wird eine neue Verbindung mit dem angeschlossenen COM-Port hergestellt und mit den entsprechenden Parametern konfiguriert. Der einzige Parameter der verändert werden muss, ist die Geschwindigkeit mit der eine Verbindung aufgebaut werden soll. Die richtige Geschwindigkeit wird immer im Namen der verwendeten Datei mitgeführt, so dass man diese im Hyperterminal einstellen kann.

1.1.6.1. uart01_02400.s43

MSP430F1232 Education System 1.3 - USART0 UART 2400 Ultra-low Power Echo ISR, 32kHz ACLK

Description; Echo a received character, RX ISR used. Normal mode is LPM3,
USART0 RX interrupt triggers TX Echo.
ACLK = UCLK0 = LFXT1 = 32768Hz, MCLK = SMCLK = DCOCLK
Baud rate divider with 32768hz XTAL @2400 = 32768Hz/2400 = 13.65 (000D 6Bh)
An external watch crystal on XIN XOUT is required for ACLK

Das Programm initialisiert das USART0 Modul als UART mit einer Bitgeschwindigkeit von 2400bit/s. Als Takt für den UART wird der ACLK (externer Quarz an XIN und XOU mit 32 768 Hz) verwendet. Aus dem ACLK wird die Bitgeschwindigkeit durch entsprechendes Teilen und Modulieren gewonnen. Sobald der UART ein komplettes Byte empfangen hat, wird der RX Empfangs-Interrupt generiert. Innerhalb der RX-Interrupt-Routine wird das empfangene Byte in das TX Senderegister kopiert. Der UART generiert dann automatisch am TX Pin das zugehörige Sendesignal. Die CPU befindet sich hauptsächlich im Low-Power-Mode, erst durch den Empfang eines Bytes und die damit verbundene Generierung des RX Interrupts wird die CPU aufgeweckt. Nach der Abarbeitung des RX-ISR kehrt die CPU wieder in den Low-Power-Modus zurück. Das Programm realisiert damit ein Echo. Das empfangene Byte wird an den Sender zurückgesendet.

```
#include "msp430x12x2.h"
;-------------------------------------------------------------------------------------------------------------------------
                ORG             0E000h                  ; Program Start
;-------------------------------------------------------------------------------------------------------------------------
RESET           mov.w           #0300h,SP               ; Initialize 'F123(2) stackpointer
                call            #Init_Sys               ;
                                ;
Mainloop        bis.b           #LPM3,SR                ; Enter LPM3
                jmp             Mainloop                ; Do nothing
                                                        ;
;-------------------------------------------------------------------------------------------------------------------------
Init_Sys; Initalize MSP430 system
;-------------------------------------------------------------------------------------------------------------------------
StopWDT         mov.w           #WDTPW+WDTHOLD,&WDTCTL  ; Stop WDT
SetupUART0      mov.b           #CHAR,&UCTL0            ; 8-bit characters
                mov.b           #SSEL0,&UTCTL0          ; UCLK = ACLK
                mov.b           #00Dh,&UBR00            ; 32k/2400 - 13.65
                mov.b           #000h,&UBR10            ; 32k 2400
                mov.b           #06Bh,&UMCTL0           ; 32k 2400 modulation
                bis.b           #UTXE0+URXE0,&ME2       ; Enable USART0 TXD/RXD
                bis.b           #URXIE0,&IE2            ; Enable USART0 RX interrupt
SetupP3         bis.b           #030h,&P3SEL            ; P3.4,5 = USART0 TXD/RXD
                bis.b           #010h,&P3DIR            ; P3.4 = output direction
                eint                                    ; General enable interrupts
                ret                                     ; Return from subroutine
                                                        ;
;-------------------------------------------------------------------------------------------------------------------------
USART0RX_ISR; Echo back RXed character, confirm TX buffer is ready first
;-------------------------------------------------------------------------------------------------------------------------
TX1             bit.b           #UTXIFG0,&IFG2          ; USART0 TX buffer ready?
                jz              TX1                     ; Jump is TX buffer not ready
                mov.b           &RXBUF0,&TXBUF0         ; TX -> RXed character
                reti                                    ;
                                                        ;
;-------------------------------------------------------------------------------------------------------------------------
; Interrupt Vectors Used MSP430x12x
;-------------------------------------------------------------------------------------------------------------------------
                ORG             0FFFEh                  ;
                DW              RESET                   ; POR, ext. Reset, Watchdog
                ORG             0FFEEh                  ;
                DW              USART0RX_ISR            ; USART0 receive
                end
```

1.1.6.2. uart02_19200.s43

MSP430F1232 Education System 1.3 - USART0 UART 19200 Ultra-low Echo ISR,
32 kHz ACLK+DCO
ACLK = LFXT1/8 = 32768/8, MCLK = SMCLK = UCLK0 = DCOCLK = 1048576 Baud rate divider with 1048576 Hz = 1048576 Hz/19200 ~ 55 (0036h)

Dieses Programm realisiert ein UART Echo. Ein empfangenes Byte wird an den Sender zurück übertragen. Der interne DCO kann mit Hilfe des externen 32 kHz Quarz so kalibriert werden, dass auf Grundlage des DCO eine Bitgeschwindigkeit von 19200 bit/s erreicht wird. Das Hauptprogramm befindet sich nach der Initialisierung im Low-Power-Mode. Das Senden, des empfangenen Bytes erfolgt in der RX Interrupt Routine.

```
Delta equ 256   ; Delta = (target DCO)/(4096) = 1048576

; #include "msp430x12x2.h
;-------------------------------------------------------------------------------------------------------------------------
                ORG             0E000h                  ; Program Start
;-------------------------------------------------------------------------------------------------------------------------
RESET           mov.w           #0300h,SP               ; Initialize stackpointer
                call            #Init_Sys               ;
                                                        ;
Mainloop        bit.b           #TXEPT,&UTCTL0          ; Confirm no TXing before --> LPM3
                jz              Mainloop                ;
                bis.b           #LPM3,SR                ; Enter LPM3
                jmp             Mainloop                ;
                                                        ;
;-------------------------------------------------------------------------------------------------------------------------
Init_Sys; Initalize MSP430 system
;-------------------------------------------------------------------------------------------------------------------------
StopWDT         mov.w           #WDTPW+WDTHOLD,&WDTCTL  ;    Stop WDT
SetupBC         bis.b           #DIVA1+DIVA0,&BCSCTL1   ; ACLK = LFXT1CLK/8
                call            #Set_DCO                ; Calibrate DCOCLK
SetupUART0      mov.b           #CHAR,&UCTL0            ; 8-bit char
                mov.b           #SSEL1+URXSE,&UTCTL0    ; UCLK = SMCLK, start edge detect
                mov.b           #036h,&UBR00            ; 1MHz 19200
                mov.b           #000h,&UBR10            ; 1MHz 19200
                mov.b           #000h,&UMCTL0           ; 1MHz 19200 modulation
                bis.b           #UTXE0+URXE0,&ME2       ; Enable USART0 TXD/RXD
                bis.b           #URXIE0,&IE2            ; Enable USART0 RX interrupt
SetupP3         bis.b           #030h,&P3SEL            ; P3.4,5 = USART0 TXD/RXD
                bis.b           #010h,&P3DIR            ; P3.4 = output direction
                eint                                    ; General enable interrupts
                ret                                     ; Return from subroutine
                                                        ;
;-------------------------------------------------------------------------------------------------------------------------
Set_DCO; Subroutine: Sets DCO to selected frequency based on Delta.
;               R14 and R15 are used, ACLK= 32768/8 Timer_A clocked by DCOCLK
;-------------------------------------------------------------------------------------------------------------------------
                clr.w           R15                     ;
Setup_TA        mov.w           #TASSEL1+TACLR+MC1,&TACTL   ;    SMCLK, clear, cont. mode
Setup_CC2       mov.w           #CCIS0+CM0+CAP,&CCTL2   ; Define CCR2,CAP,ACLK
```

```
Test_DCO    bit.w    #CCIFG,&CCTL2          ; Test capture flag
            jz       Test_DCO               ;
            bic.w    #CCIFG,&CCTL2          ; Clear capture flag
                                            ;
AdjDCO      mov.w    &CCR2,R14              ; R14 = captured SMCLK
            sub.w    R15,R14                ; R14 = capture difference
            mov.w    &CCR2,R15              ; R15 = captured SMCLK
            cmp.w    #Delta,R14             ; Delta = SMCLK/(32768/4)
            jlo      IncDCO                 ;
            jeq      DoneDCO                ;
DecDCO      dec.b    &DCOCTL                ; Slow DCO with DCO and MOD
            jnz      Test_DCO               ; Slower?
            bit.b    #07h,&BCSCTL1          ; Can RSEL.x be decremented?
            jz       DoneDCO                ; jmp>DCO at slowest setting
            dec.b    &BCSCTL1               ; Decrement RSEL.x
            jmp      Test_DCO               ;
IncDCO      inc.b    &DCOCTL                ; Speed DCO with DCO and MOD
            jnc      Test_DCO               ; Faster?
            cmp.b    #07h,&BCSCTL1          ; Can RSEL.x be increased?
            jz       DoneDCO                ; jmp> DCO at fastest settting
            inc.b    &BCSCTL1               ; Increment RSEL.x
            jmp      Test_DCO               ;
DoneDCO     clr.w    &CCTL2                 ; Stop CCR2
            clr.w    &TACTL                 ; Stop timer_A
            ret                             ; Return from subroutine
                                            ;
;-------------------------------------------------------------------------------
USART0RX_ISR; Echo back RXed character, confirm TX buffer is ready first
;-------------------------------------------------------------------------------
            bit.b    #URXIFG0,&IFG2         ; Interrupt from complete char?
            jc       TX1                    ; Jump--> interrupt from char
            bic.b    #SCG1+SCG0,0(SP)       ; Enter LPM0 on reti
            reti                            ;
                                            ;
TX1         bit.b    #UTXIFG0,&IFG2         ; USART0 TX buffer ready?
            jz       TX1                    ; Jump is TX buffer not ready
            mov.b    &RXBUF0,&TXBUF0        ; TX -> RXed character
            bic.b    #LPM3,0(SP)            ; Exit LPM3 on reti
            reti                            ;
                                            ;
;-------------------------------------------------------------------------------
; Interrupt Vectors Used MSP430x12x
;-------------------------------------------------------------------------------
            ORG      0FFFEh                 ;
            DW       RESET                  ; POR, ext. Reset, Watchdog
            ORG      0FFEEh                 ;
            DW       USART0RX_ISR           ; USART0 receive
            end
```

1.1.6.3. uart03_19200.s43

MSP430F1232 Education System 1.3 - USART0 UART 19200 Echo ISR, 32kHz ACLK+DCO

ACLK = LFXT1/8 = 32768/8, MCLK = SMCLK = UCLK0 = DCOCLK = 1048576 Baud rate divider with 1048576 Hz = 1048576 Hz/19200 ~ 55 (0036h)
An external 32 kHz watch crystal on XIN XOUT is required for ACLK

Dieses Programm realisiert ein UART Echo. Ein empfangenes Byte wird an den Sender zurück geschickt. Der interne DCO kann mit Hilfe des externen 32 kHz Quarz so kalibriert werden, dass auf Grundlage des DCO eine Bitgeschwindigkeit von 19200 Bit/s erreicht wird. Das Hauptprogramm befindet sich nach der Initialisierung im Low-Power-Mode. Das Senden des empfangenen Bytes erfolgt in der RX Interrupt Routine. Im Gegensatz zu der Lösung zwei wird dieses nicht im Ultra-Lowpower-Mode betrieben.

```
Delta   equ   256    ; Delta = (target DCO)/(4096) = 1048576

#include "msp430x12x2.h"
;-------------------------------------------------------------------------------------------------------------------------
                ORG             0E000h                  ; Program Start
;-------------------------------------------------------------------------------------------------------------------------
RESET           mov.w           #300h,SP                ; Initialize stackpointer
                call            #Init_Sys               ;
                                                        ;
Mainloop        bis.b           #LPM0,SR                ; Enter LPM0
                nop                                     ; Needed only for C-spy
                                                        ;
;-------------------------------------------------------------------------------------------------------------------------
Init_Sys; Initalize MSP430 system
;-------------------------------------------------------------------------------------------------------------------------
StopWDT         mov.w           #WDTPW+WDTHOLD,&WDTCTL      ; Stop WDT
SetupBC         bis.b           #DIVA1+DIVA0,&BCSCTL1       ; ACLK = LFXT1CLK/8
Setup_TA        mov.w           #TASSEL1+MC1,&TACTL         ; SMCLK, Continous Mode
Setup_CC2       mov.w           #CCIS0+CM0+CAP+CCIE,&CCTL2  ; CAP,ACLK, with interrupt
SetupUART0      mov.b           #CHAR,&UCTL0                ; 8-bit char
                mov.b           #SSEL1,&UTCTL0              ; UCLK = SMCLK
                mov.b           #036h,&UBR00                ; 1MHz 19200
                mov.b           #000h,&UBR10                ; 1MHz 19200
                mov.b           #000h,&UMCTL0               ; 1MHz 19200 modulation
                bis.b           #UTXE0+URXE0,&ME2           ; Enable USART0 TXD/RXD
                bis.b           #URXIE0,&IE2                ; Enable USART0 RX interrupt
SetupP3         bis.b           #030h,&P3SEL                ; P3.4,5 = USART0 TXD/RXD
                bis.b           #010h,&P3DIR                ; P3.4 = output direction
                eint                                        ; General enable interrupts
                ret                                         ; Return from subroutine
                                                            ;
;-------------------------------------------------------------------------------------------------------------------------
USART0RX_ISR; Echo back RXed character, confirm TX buffer is ready first
;-------------------------------------------------------------------------------------------------------------------------
TX1             bit.b           #UTXIFG0,&IFG2              ; USART0 TX buffer ready?
                jz              TX1                         ; Jump is TX buffer not ready
```

```
                mov.b   &RXBUF0,&TXBUF0         ; TX -> RXed character
                reti                            ;
                                                ;
;-------------------------------------------------------------------------------
TA2_ISR
;-------------------------------------------------------------------------------
AdjDCO          push.w  &CCR2                   ;
                sub.w   R15,&CCR2               ;
                cmp.w   #Delta,&CCR2            ; Delta = SMCLK/(32768/8)
                pop.w   R15                     ;
                jlo     IncDCO                  ;
                jeq     DoneDCO                 ;
DecDCO          dec.b   &DCOCTL                 ; Slow DCO with DCO and MOD
                jnz     DoneDCO                 ; Slower?
                bit.b   #07h,&BCSCTL1           ; Can RSEL.x be decremented?
                jz      DoneDCO                 ; jump>DCO at slowest setting
                dec.b   &BCSCTL1                ; Decrement RSEL.x
                reti                            ; Return from interrrupt
IncDCO          inc.b   &DCOCTL                 ; Speed DCO with DCO and MOD
                jnc     DoneDCO                 ; Faster?
                cmp.b   #07h,&BCSCTL1           ; Can RSEL.x be increased?
                jz      DoneDCO                 ; jump> DCO at fastest settting
                inc.b   &BCSCTL1                ; Increment RSEL.x
DoneDCO         reti                            ; Return from interrrupt
                                                ;
;-------------------------------------------------------------------------------
TAX_ISR; Common ISR for CCR1-4 and overflow
;-------------------------------------------------------------------------------
                add.w   &TAIV,PC                ; Add Timer_A offset vector
                reti                            ; CCR0 - no source
                reti                            ; CCR1
                jmp     TA2_ISR                 ; CCR2
;               reti                            ; CCR3
;               reti                            ; CCR4
;               reti                            ; Return from overflow ISR
                                                ;
;-------------------------------------------------------------------------------
; Interrupt Vectors Used MSP430x12x(2)
;-------------------------------------------------------------------------------
                ORG     0FFFEh                  ;
                DW      RESET                   ; POR, ext. Reset, Watchdog
                ORG     0FFEEh                  ;
                DW      USART0RX_ISR            ; USART0 receive
                ORG     0FFF0h                  ; Timer_AX Vector
                DW      TAX_ISR                 ;
                end
```

1.2. C-Beispielprogramme

In den folgenden Abschnitten werden einzelne Projekte beschrieben und verschiedene Bespielprogramme dokumentiert. Diese wurden komplett in der Hochsprache C geschrieben. Durch den Einsatz der Hochsprache C können komplexe Programmabläufe relativ einfach realisiert und übersichtlicher programmiert werden. In den meisten Fällen wird der Einsatz dieser Programmiersprache als sehr arbeitserleichternd empfunden und spart somit sehr viel Zeit während der Programmierung. Der Programmierer kann sich mehr auf das zu lösende Problem konzentrieren und wird somit nicht durch andere Probleme von der eigentlichen Arbeit abgelenkt.

Die Programmiersprache C hat in der Vergangeheit Assembler verdrängt und wird heute als Standardprogrammiersprache bevorzugt eingesetzt. Assembler wird heute nur noch dort eingesetzt, wo ein sehr kompakter Programmcode benötigt wird. Damit zeigt sich der noch einzige Nachteil beim Einsatz dieser Programmiersprache. Da diese eine Hochsprache darstellt, wird natürlich ein sehr effizienter C-Compiler vorausgesetzt.

Wie bei den Assembler-Programmen stellen die Bezeichnungen gleichzeitig den Namen des zugehörigen Ordners dar und die in Klammern stehenden Bezeichnungen sind ebenfalls die zugehörigen Projektnamen. Alle Projekte tragen den gleichen Namen wie die Assemblerprojekte, um besser Vergleiche zwischen beiden Programmiersprachen durchführen zu können. Es wurde versucht, jeweils die gleiche Funktionalität mit einem C-Programm zu realisieren. Kleine Abweichungen können hierbei natürlich nicht ausgeschlossen werden. Nachfolgend nun alle C-Programme und Projekte mit zugehöriger Beschreibung.

1.2.1. Project_1_(LED), Verwenden der acht Leuchtdioden an Port 1

In diesem Projekt werden unterschiedliche Funktionsweisen der Leuchtdioden am Port 1 demonstriert. Der Port 1 ist in dem Fall als Ausgang definiert um Daten an die Low-aktiven Leuchtdioden D0 – D7 auszugeben. Die entsprechenden Port Register müssen die Pins als Ausgang schalten. Bei den LEDs gilt, bei aktivem Debug-Fenster werden nur die unteren vier Leuchtdioden angezeigt, da an P1.4 bis P1.7 die Debug-Schnittstelle angeschlossen ist. Schliesst man das Debug-Fenster, werden alle LEDs mit ihrer festgelegten Funktionalität arbeiten.

1.2.1.1. LED_1.c

MSP430F1232 Education System 1.3 - different ways to change output levels

Die LEDs an P1.0 und P1.1 werden abwechselnd an- und ausgeschaltet. Der Unterschied besteht darin, dass im Falle von P1.0 der Pin Ausgangspegel explizit auf 1 oder 0 gesetzt wird. Im Falle von P1.1 erfolgt das An- und Ausschalten durch ein bitweises XOR (exklusives ODER). Die Warteschleife zwischen dem Umschalten der Ausgangspegel wurde in die Funktion 'delay' ausgelagert.

```
#include  <msp430x12x2.h>

// Function prototypes
void delay(unsigned int i);                 // software delay

void main(void)
{
  WDTCTL = WDTPW + WDTHOLD;                 // stop watchdog timer

  P1DIR = BIT0+BIT1;                        // P1.0 and P1.1 as output
  P1OUT = 0xFF;                             // LEDs off

  while(1)                                  // repeat forever
  {
    P1OUT |= BIT0;                          // set P1.0 (LED off)
    P1OUT ^= BIT1;                          // toggle P1.1 (LED on)
    delay(30000);                           // call delay subroutine
    P1OUT &= ~BIT0;                         // clear P1.0
    P1OUT ^= BIT1;                          // toggle P1.1 (LED off)
    delay(30000);                           // call delay subroutine
  }
}

void delay(unsigned int i)
{
                        // Note: i is an unsigned integer. If not declared unsigned, 65000 in 16 bits
```

```
                        // becomes a negative number, and the loop is executed only once!
  while(i > 0) i--;
}
```

1.2.1.2. LED_2.c

MSP430F1232 Education System 1.3 - LED Knight Rider

Dieses Programm dient zur Demonstration der C-Schiebebefehle durch Ausgabe verschiedener Bitmuster auf den LEDs an Port1. Die Funktion 'led_right_shift' geht von einem Bitmuster 0xFF aus. Das Bitmuster wird jeweils um ein Bit arithmetisch nach rechts geschoben und anschließend ins Port1 Ausgaberegister geschoben. Die Schleife der Funktion endet, sobald das Bitmuster den Wert 0 erreicht hat. Dadurch wird erreicht, dass von links nach rechts nacheinander die LEDs angeschaltet werden, bis zum Ende der Funktion alle LEDs leuchten. Die Funktion 'led_left_shift' macht genau das Gleiche, jedoch von rechts nach links.

BSP für 'led_right_shift':
bitmap
11111111
01111111
00111111
00011111
00001111
00000111
00000011
00000001
00000000

```
#include  <msp430x12x2.h>

// Function prototypes
void delay(unsigned int i);                 // Software delay
void led_right_shift(unsigned int i);       // right shift
void led_left_shift(unsigned int i);        // left shift

void main(void)
{
  WDTCTL = WDTPW + WDTHOLD;                 // Stop watchdog timer

  P1DIR = 0xFF;                             // P1 as output
  P1OUT = 0xFF;                             // LEDs off

  while(1) {
    led_right_shift(20000);                 // call subroutine
    led_left_shift(20000);                  // call subroutine
  }
}

void led_right_shift(unsigned int i) {
  unsigned char bitmap;                     // decl. local variable
  bitmap = 0xFF;
  while(bitmap > 0)                         // do until bitmap = 0
  {
    P1OUT = bitmap;                         // write bitmap to output register
    delay(i);                               // wait
```

```
    bitmap = bitmap >> 1;                    // right shift arithmetically
  }
  P1OUT = bitmap;                            // write bitmap to output register
  delay(i);                                  // wait
}

void led_left_shift(unsigned int i) {
  unsigned char bitmap;                      // decl. local variable
  bitmap = 0xFF;
  while(bitmap > 0)                          // do until bitmap = 0
  {
    P1OUT = bitmap;                          // write bitmap to output register
    delay(i);                                // wait
    bitmap = bitmap << 1;                    // left shift arithmetically
  }
  P1OUT = bitmap;                            // write bitmap to output register
  delay(i);
}

void delay(unsigned int i)
{

                          // Note: i is an unsigned integer. If not declared unsigned, 65000 in 16 bits
                          // becomes a negative number, and the loop is executed only once!
   while(i > 0) i--;
}
```

1.2.2. Project_2_(Taster)

In diesem Projekt wird die Tasterfunktion mit Hilfe der Interrupts des MSP430F1232 demonstriert. Die genaue Funktionsweise und Belegung der Taster findet man im Handbuch des MSP430 Education System.

1.2.2.1. Taster.c

MSP430F1232 Education System 1.3 - Button Demo

Alle Pins an Port 1 sind als Ausgang geschaltet. Für die Pins (Port 2), an denen Taster angeschlossen sind, wird die Generierung von Interrupts erlaubt. Danach geht der MSP430 in den Low-Power-Mode. Durch Betätigung eines Tasters wird der Controller aufgeweckt und generiert den Port 2 Interrupt. Innerhalb der Interrupt-Routine wird zuerst, durch Aufruf der Subroutine DebounceDelay, eine gewisse Zeit gewartet. Dies soll der Taster-Entprellung dienen. Anschließend wird gestestet, welcher Eingangsport (und damit verbunden, welcher Taster) den Interrupt generiert hat (bzw. welcher Taster gedrückt wurde) Entsprechend wird dann der Ausgangspegel eines Port1 Pins getoggelt und die entsprechende zugehörige LED ein- bzw. ausgeschaltet.

```
#include  <msp430x12x2.h>

                                        // Function prototypes
void DebounceDelay(void);               // Software delay

void main(void)
{
  WDTCTL = WDTPW + WDTHOLD;             // Stop watchdog timer

  P1DIR = 0xFF;                         // P1.0-P1.7 as output
  P2IE = BIT0+BIT1+BIT2+BIT5;           // enable interrupt for P2 input pins
  _EINT();                              // enable general interrupt

  while(1)                              // Repeat forever
  {
  }
}

                                        // interrupt service routine for Port2
interrupt[PORT2_VECTOR] void P2_ISR(void) {
   DebounceDelay();                     // call delay subroutine
   if(P2IFG & BIT0) {                   // P2.0 caused the interrupt?
     P1OUT ^= BIT0; P2IFG &= ~ BIT0;    // yes, toggle P1.0 and reset P2.0 interrupt flag
   } else if(P2IFG & BIT1) {            // or P2.1 caused the interrupt?
     P1OUT ^= BIT1; P2IFG &= ~ BIT1;    // yes, toggle P1.1 and reset P2.1 interrupt flag
   } else if(P2IFG & BIT2) {            // or P2.2 caused the interrupt?
     P1OUT ^= BIT2; P2IFG &= ~ BIT2;    // yes, toggle P1.2 and reset P2.2 interrupt flag
   } else if(P2IFG & BIT5) {            // or P2.5 caused the interrupt?
```

```
    P1OUT ^= BIT3; P2IFG &= ~ BIT5;          // yes, toggle P1.3 and reset P2.5 interrupt flag
  }
}

void DebounceDelay(void)
{
                          // Note: i is an unsigned integer. If not declared unsigned, 65000 in 16 bits
                          // becomes a negative number, and the loop is executed only once!
  unsigned int i;
  for (i = 50000; i > 0; i--);
}
```

1.2.3. Project_3_(ADC10)

In diesem Projekt wird die Funktion des internen A/D-Wandlers an Port 3 demonstriert. Die angeschlossenen Potentiometer stellen eine Spannung an dem jeweiligen Port bereit, die der A/D-Wandler dann in einen digitalen Wert umwandelt und vom Mikrocontroller weiter verarbeitet werden kann. Als Ausgabebaugruppen werden die LEDs an Port 1 sowie das an Port 2 angeschlossene analoge Drehspulanzeigenelement verwendet. In diesem Projekt können drei unterschiedliche Programme eingefügt werden, in denen auf unterschiedlichste Art und Weise die Funktion des internen ADC demonstriert wird.

1.2.3.1. ADC10_1.c

MSP430F1232 Education System 1.3 - ADC10 Demo (1)

AD-Wandlung des Spannungspegels an A6

Die Samplefrequenz wird getriggert durch den CCR0 Interrupt. Innerhalb des CCR0 Interrupt wird eine neue AD-Wandlung initialisiert und freigegeben. Das Triggern des ADC10 Sampling-Timers erfolgt durch den CCR2 Block. Sobald der AD-Wandler mit der Wandlung fertig ist, wird seine Interrupt Routine 'ADC10_ISR' ausgeführt. Darin wird mit einem ausgeklügelten Algorithmus der Wandlungswert als LED-Balken dargestellt. Der Wandlungswert wird zudem verwendet, um den Duty Cycle der CCR1 PWM zu verändern. Das CCR1 pulsmodulierte Signal ist auf das analoge Anzeigeninstrument geschaltet.

```
#define   PWM_Offset   10
#include  <msp430x12x2.h>

// Function prototypes

void main(void)
{
  WDTCTL = WDTPW + WDTHOLD;             // Stop watchdog timer
          // setup ADC
  ADC10CTL1 = INCH_6+SHS_3;             // P3.6, TA2 trigger sample start
  ADC10AE |= BIT6;                      // P3.6 ADC10 option select
           // setup P1
  P1DIR = 0xFF;                         // P1.0-P1.7 output
          // setup P2
  P2DIR |= BIT3;
  P2SEL |= BIT3;
          // setup CCR0
  CCTL0 = CCIE;                         // Enable interrupt
  CCR0 = 0x3FF+PWM_Offset;
          // setup CCR1
  CCTL1 = OUTMOD_7;                     // CCR1 reset/set
  CCR1 = 1;                             // CCR1 PWM Duty Cycle
```

```
        // setup CCR2
  CCTL2 = OUTMOD_3;                     // CCR2 set/reset
  CCR2 = 2;                             // CCR2 PWM Duty Cycle
        // setup Timer A
  TACTL = TASSEL0+MC0;                  // ACLK, up mode
  _EINT();
  while(1);
        //_BIS_SR(CPUOFF + GIE);        // go to sleep
}

interrupt[ADC10_VECTOR] void ADC10_ISR(void) {
   unsigned int adc_ref,bitmap;
   ADC10CTL0 &= ~ENC;                   // ADC10 disabled
   ADC10CTL0 = 0;                       // ADC10, Vref disabled completely

   adc_ref = 0x7EF;
   bitmap = 0x1FF;
   while(bitmap > 0 && adc_ref > ADC10MEM) {
    bitmap = bitmap >> 1;
    adc_ref = adc_ref >> 1;
   }
   P1OUT = bitmap;
   CCR1 = ADC10MEM+PWM_Offset;
}

interrupt[TIMERA0_VECTOR] void TA0_ISR(void) {
   ADC10CTL0 = SREF_0+ADC10SHT_2+REFON+ADC10ON+ADC10IE;
   ADC10CTL0 |= ENC;
}
```

1.2.3.2. ADC10_2.c

MSP430F1232 Education System 1.3 - ADC10 Demo (2)

AD-Wandlung des Spannungspegels an A7

Die Samplefrequenz wird getriggert durch den CCR0 Interrupt. Innerhalb des CCR0 Interrupt wird dann eine neue AD-Wandlung initialisiert und freigegeben. Das Triggern des ADC10 Sampling Timers erfolgt durch den CCR2 Block. Sobald der AD-Wandler mit der Wandlung fertig ist, wird seine Interrupt Routine 'ADC10_ISR' ausgfeführt. Es wird kontrolliert, ob der Wandlungswert über oder unter einem bestimmten Schwellwert liegt und entsprechend die LED an P1.0 an- oder ausgeschaltet. Anschließend wird der Wandlungswert verwendet, um den Duty Cycle der CCR1 PWM zu verändern. Das CCR1 pulsmodulierte Signal ist auf das analoge Anzeigeninstrument geschaltet.

```
#define PWM_Offset 10
#include <msp430x12x2.h>

// Function prototypes

void main(void)
{
  WDTCTL = WDTPW + WDTHOLD;             // Stop watchdog timer
        // setup ADC
  ADC10CTL1 = INCH_7+SHS_3;             // P3.7, TA2 trigger sample start
  ADC10AE |= BIT7;                      // P3.7 ADC10 option select
        // setup P1
  P1DIR = 0xFF;                         // P1.0-P1.7 output
  P1OUT = 0xFF;
        // setup P2
  P2DIR |= BIT3;
  P2SEL |= BIT3;
        // setup CCR0
  CCTL0 = CCIE;                         // Enable interrupt
  CCR0 = 0x3FF+PWM_Offset;
        // setup CCR1
  CCTL1 = OUTMOD_7;                     // CCR1 reset/set
  CCR1 = 1;                             // CCR1 PWM Duty Cycle
        // setup CCR2
  CCTL2 = OUTMOD_3;                     // CCR2 set/reset
  CCR2 = 2;                             // CCR2 PWM Duty Cycle
        // setup Timer A
  TACTL = TASSEL0+MC0;                  // ACLK, up mode
  _EINT();
  while(1);
        //_BIS_SR(CPUOFF + GIE);        // go to sleep
}

interrupt[ADC10_VECTOR] void ADC10_ISR(void) {
   unsigned int adc_ref;
   ADC10CTL0 &= ~ENC;                   // ADC10 disabled
```

```
  ADC10CTL0 = 0;                          // ADC10, Vref disabled completely

  adc_ref = 0x1FF;
  P1OUT &= ~BIT0;
  if(ADC10MEM > adc_ref)
    P1OUT |= BIT0;
  CCR1 = ADC10MEM+PWM_Offset;
}

interrupt[TIMERA0_VECTOR] void TA0_ISR(void) {
  ADC10CTL0 = SREF_0+ADC10SHT_2+REFON+ADC10ON+ADC10IE;
  ADC10CTL0 |= ENC;
}
```

1.2.3.3. ADC10_3.c

MSP430F1232 Education System 1.3 - ADC10 Demo (3)

AD-Wandlung des Spannungspegels an A7

Die Samplefrequenz wird getriggert durch den CCR0 Interrupt. Innerhalb des CCR0 Interrupt wird eine neue AD-Wandlung initialisiert und freigegeben. Das Triggern des ADC10 Sampling Timers erfolgt durch den CCR2 Block. Sobald der AD-Wandler mit der Wandlung fertig ist, wird seine Interrupt Routine 'ADC10_ISR' ausgeführt. Darin wird mit einem ausgeklügelten Algorithmus der Wandlungswert als LED-Balken dargestellt. Der Wandlungswert wird zudem verwendet, um den Duty Cycle der CCR1 PWM zu verändern. Das CCR1 pulsmodulierte Signal ist auf das analoge Anzeigegerät geschaltet.
In diesem Programm wird der Wert zusätzlich mit einem Offset belegt, so dass der Ausgabewert zwischen 1FF und 7EF liegt.

```
#define PWM_Offset 10

#include  <msp430x12x2.h>

// Function prototypes

void main(void)
{
  WDTCTL = WDTPW + WDTHOLD;             // Stop watchdog timer
        // setup ADC
  ADC10CTL1 = INCH_7+SHS_3;             // P3.7, TA2 trigger sample start
  ADC10AE |= BIT7;                      // P3.7 ADC10 option select
        // setup P1
  P1DIR = 0xFF;                         // P1.0-P1.7 output
        // setup P2
  P2DIR |= BIT3;
  P2SEL |= BIT3;
        // setup CCR0
  CCTL0 = CCIE;                         // Enable interrupt
  CCR0 = 0x3FF+PWM_Offset;
        // setup CCR1
  CCTL1 = OUTMOD_7;                     // CCR1 reset/set
  CCR1 = 1;                             // CCR1 PWM Duty Cycle
        // setup CCR2
  CCTL2 = OUTMOD_3;                     // CCR2 set/reset
  CCR2 = 2;                             // CCR2 PWM Duty Cycle
        // setup Timer A
  TACTL = TASSEL0+MC0;                  // ACLK, up mode
  _EINT();
  while(1);
        //_BIS_SR(CPUOFF + GIE);        // go to sleep
}
```

```
interrupt[ADC10_VECTOR] void ADC10_ISR(void) {
  unsigned int adc_ref,bitmap;
  ADC10CTL0 &= ~ENC;                    // ADC10 disabled
  ADC10CTL0 = 0;                        // ADC10, Vref disabled completely

  adc_ref = 0x7EF;
  bitmap = 0x1FF;
  while(bitmap > 0 && adc_ref > ADC10MEM) {
   bitmap = bitmap >> 1;
   adc_ref = adc_ref >> 1;
  }
  P1OUT = bitmap;
  CCR1 = ADC10MEM+PWM_Offset;
}

interrupt[TIMERA0_VECTOR] void TA0_ISR(void) {
  ADC10CTL0 = SREF_0+ADC10SHT_2+REFON+ADC10ON+ADC10IE;
  ADC10CTL0 |= ENC;
}
```

1.2.4. Project_4_(TimerA)

Der Timer ist eine sehr vielseitig einsetzbare, interne Baugruppe des MSP430F1232. Mit dem Timer können verschiedene Zeiten, zur späteren Verwendung in einem eigenen Programm, generiert werden. Dadurch ist es möglich, verschiedene Programme mit einer einheitlichen Zeitbasis zu programmieren. Timer sind ebenfalls zur Generierung von Tönen sehr wichtig, da durch ihre Verwendung Pulsweiten sehr einfach eingestellt werden können, die später im Programm weiter verarbeitet werden können.
Die nachfolgenden Programme zeigen die Verwendung dieser Baugruppe an Hand des Lautsprechers an Port 3 sowie der LEDs an Port 1 des MSP430 Education System.

1.2.4.1. TimerA_1.c

MSP430F1232 Education System 1.3 - Timer_A Toggle P1.0-3 CCRx Contmode ISR, ACLK

Dieses Programm soll die Verwendung der Interrupt-Quellen des Timer_A-Moduls veranschaulichen. Timer_A wird im Continous-Mode betrieben. Die Ausgabeeinheiten der CCRx Blöcke werden auf toggeln eingestellt und für die zugehörigen Port1 Ausgangspins die Modul Funktionalität ausgewählt. Durch Verwendung der Ausgabeeinheiten der CCRx Blöcke, erfolgt das Toggeln der Ausgangspegel hardwareseitig und unabhängig von der Software. Nachdem auf der Platine an Port1 LEDs angebracht sind, wird zudem das Toggeln der Ausgänge visualisiert.

```
#include  "msp430x12x.h"

void main(void)
{
  unsigned int i;
  WDTCTL = WDTPW + WDTHOLD;             // Stop WDT

  TACTL = TASSEL0 + TACLR + TAIE;       // ACLK, clear TAR, interrupt enabled
  CCTL0 = OUTMOD_4 + CCIE;              // CCR0 toggle, interrupt enabled
  CCTL1 = OUTMOD_4 + CCIE;              // CCR1 toggle, interrupt enabled
  CCTL2 = OUTMOD_4 + CCIE;              // CCR2 toggle, interrupt enabled
  CCR0 = 0x7FFF;
  CCR1 = 0x3FFF;
  CCR2 = 0x1FFF;
  P1SEL |= 0x0E;                        // P1.1 - P1.3 option select
  P1DIR |= 0x0F;                        // P1.0 - P1.3 outputs
  TACTL |= MC1;                         // Start Timer_A in continuous mode
  _EINT();                              // Enable interrupts

  for (;;)
  {
    _BIS_SR(CPUOFF);                    // CPU off
```

```
    _NOP();                                 // Required only for C-spy
  }
}

          // Timer A0 interrupt service routine
interrupt[TIMERA0_VECTOR] void Timer_A0 (void)
{
  CCR0 += 0x7FFF;                           // Add Offset to CCR0
}

          // Timer_A3 Interrupt Vector (TAIV) handler
interrupt [TIMERA1_VECTOR] void Timer_A1(void)
{
  switch( TAIV )
  {
  case  2: CCR1 += 0x3FFF;                  // Add Offset to CCR1
          break;
  case  4: CCR2 += 0x1FFF;                  // Add Offset to CCR2
          break;
  case 10: P1OUT ^= 0x01;                   // Timer_A3 overflow
          break;
 }
}
```

1.2.5. Project_5_(TimerA_PWM)

In diesem Projekt wird ebenfalls der Timer_A verwendet, hier jedoch zum Generieren eines pulsweitenmodulierten Signals. Durch den Einsatz des Timers gelingt es, wie schon bei dem Lautsprecher, sehr schnell eine gewünschte Pulsweite einzustellen.

1.2.5.1. TimerA_PWM.c

MSP430F1232 Education System 1.3 - Timer_A Pulse Width Modulation Demo

Description; This program will generate a two PWM outputs on P1.2/1.3 using

- Timer_A in an upmode. The value in CCR0, 30000, defines the period and the values in CCR1 and CCR2 the duty PWM cycles.
- Using 32 kHz ACLK as TACLK,
- the timer period is 15.6ms with a 75% duty cycle on P1.2 and 25% on P1.3.
- Normal mode LPM3

Dieses Programm zeigt die einfachste Variante einer PWM. Timer_A wird im up-mode betrieben, d.h. er zählt bis zum Wert, der im Register CCR0 gespeichert ist und beginnt dann wieder bei 0. Die Ausgabeeinheiten der Module CCR1 und CCR2 werden auf reset/set initialisiert. Die Werte im CCR1 und CCR2 Register bestimmen die Pulsweite der Ausgangssignale, wobei die Periodendauer durch den Wert im CCR0 Register und die Taktung des Timer_A (ACLK) bestimmt wird. Im Beispiel wird eine 75% und 25% Pulsdauer PWM erzeugt. Die CPU befindet sich im Low-Power-Mode. Die PWM erfolgt rein durch externe Taktung des Timer_A Moduls mit dem 32kHz Quarz.

```
#include <msp430x12x.h>

void main(void)
{
  WDTCTL = WDTPW +WDTHOLD;              // Stop WDT
  TACTL = TASSEL0 + TACLR;              // ACLK, Clear Tar
  CCTL0 = OUTMOD_4;
  CCR0 = 30000;                         // PWM Period
  CCTL1 = OUTMOD_7;                     // CCR1 reset/set
  CCR1 = 22500;                         // CCR1 PWM duty cycle
  CCTL2 = OUTMOD_7;                     // CCR2 reset/set
  CCR2 = 7500;                          // CCR2 PWM duty cycle
  P1DIR |= BIT1+BIT2+BIT3;              // P1.2 and P1.3 output
  P1SEL |= BIT1+BIT2+BIT3;              // P1.2 and P1.3 TA1/2 otions
  TACTL |= MC0;                         // Start Timer_A in up mode

  for (;;)
  {
```

```
    _BIS_SR(CPUOFF);                        // Enter LPM0
    _NOP();                                 // Required only for C-spy
  }
}
```

1.2.6. Project_6_(LCD)

In diesem Projekt wird das richtige Verwenden des LC-Displays demonstriert. Die Initialisierung und alle LCD-Funktionen wurden in einem eigenen File geschrieben, um diese problemlos weiter verwenden zu können. Im MSP430 Education System Handbuch wurden alle Parameter und Funktionen zur richtigen Initialisierung des Displays beschrieben, können aber auch aus dem LCD-Datenblatt entnommen werden.

1.2.6.1. LCD_1.c

MSP430F1232 Education System 1.3 - LCD Demo

Dieses Programm veranschaulicht den grundlegenden Einsatz des Software-Moduls 'lcd.c' zur Ansteuerung und Ausgabe von Zeichen auf dem LCD. Das Software-Modul verwendet dazu einige User-Funktionen. Die Wichtigsten hiervon werden in diesem Programm verwendet. Zur genaueren Beschreibung der einzelnen Funktionen siehe: lcd.c

```
#include  <msp430x12x2.h>
#include  "lcd.c"

unsigned char string1[] = "Hallo! Ich bin";
unsigned char string2[] = "ein MSP430F1232";

void main(void)
{
  WDTCTL = WDTPW + WDTHOLD;             // Stop watchdog timer
  lcd_init(16);                         // LCD init
  lcd_clear();                          // clear all LCD digits, cursor to postion 0,0
  lcd_puts(string1);                    // print string to current cursor position
  lcd_gotoxy(0,1);                      // move cursor to position 0,1 (new line)
  lcd_puts(string2);                    // print string to current cursor position

  while(1)                              // Repeat forever
  {
  }
}
```

1.2.6.2. lcd.c

```
//*******************************************************************************************************************
//
//                lcd.c - Program
//
//  Description: LCD-Functions (4-bit Mode)
//
//  Pinbelegung LCD:
//--------------------
//  1 GND- 9  GND
//  2 +5V- 10 VCC
//  3 VLC- LCD HEADER Vo
//  4 RS - 1  Register Select
//  5 RD - 2  Read/write
//  6 EN - 3  LCD_Enable
//  7 D0 - 5  PA0 - data bus  (not used in 4bit-mode)
//  8 D1 - 6  PA0 - data bus  (not used in 4bit-mode)
//  9 D2 - 7  PA0 - data bus  (not used in 4bit-mode)
// 10 D3 - 8  PA0 - data bus  (not used in 4bit-mode)
// 11 D4 - 5  PA0 - data bus
// 12 D5 - 6  PA0 - data bus
// 13 D6 - 7  PA0 - data bus
// 14 D7 - 8  PA0 - data bus
//*******************************************************************************************************************

#include <msp430x12x2.h>
#include "lcd.h"

unsigned int _base_y[4] = {0x80, 0xc0, 0x94, 0xd4};                    // example for 4x20 display
        // 1. line 0x00..0x13 + Bit7==0x80
        // 2. line 0x40..0x53 + Bit7==0xc0
        // 3. line 0x14..0x27 + Bit7==0x94
        // 4. line 0x54..0x67 + Bit7==0xd4
        // BIT7 nessesary for command "Set DD RAM address"

unsigned int _lcd_x,_lcd_y,_lcd_maxx;



#define LCD_DATA_OUT            P1OUT
#define LCD_DATA_DIR            P1DIR
#define LCD_COMM_OUT            P3OUT
#define LCD_COMM_DIR            P3DIR
#define DATA_LINE_BITMASK    ( 0x000F )                                // LCD D7 - D4
#define CONTROL_LINE_BITMASK ( 0x0003 )                                // bit0 -> RS | bit1 -> E

#define RS_BIT              ( 0x0001 )              // RS Display
#define ENABLE_BIT          ( 0x0002 )              // Enable Display



#define SELECT_COMMAND_REG   LCD_COMM_OUT &= ~RS_BIT                   // set RS to low
#define SELECT_DATA_REG      LCD_COMM_OUT |=  RS_BIT                   // set RS high


// Read-mode wird bis auf weiteres nicht verwendet
//#define DISPLAY_WRITE       LCD_COMM_OUT &= ~RW_BIT                  // write to display (low)
//#define DISPLAY_READ        LCD_COMM_OUT |=  RW_BIT                  // read from display (high)
```

```
// "Taktleitung" des LCD
#define DISPLAY_ENABLE_HIGH  LCD_COMM_OUT |=  ENABLE_BIT          // enable HIGH
#define DISPLAY_ENABLE_LOW   LCD_COMM_OUT &= ~ENABLE_BIT          // enable LOW


// command und data- Makro wird für Aufruf der Funktion
// _SendByteToLCD_4BIT benötigt
#define COMMAND      0
#define DATA         1



//*****************************************************************************************************************
// Description  : Waiting for Busy-flag
// Input        : none
// Output       : none
//*****************************************************************************************************************
void lcd_delay(void)
{
unsigned int count;
// "2x16, 4x16 and 4x20 display works with 40µsec delay"  (Udo)


   // delay loop
   for (count=0; count<50; count++);
}



//*****************************************************************************************************************
// Description  : Sends byte to lcd in 4bit-mode
// Input        : data - data to send
//              : reg - specifies the lcd-register
// Output       : none
//*****************************************************************************************************************
void _SendByteToLCD_4BIT(unsigned char data, unsigned char reg)
{
// Daten sind gültig bei HIGH-LOW Flanke des enable-Signals !!
// High nibble (BIT7-BIT4) are transfered first

   lcd_delay();


   if (reg == COMMAND)
      SELECT_COMMAND_REG;
   else
      SELECT_DATA_REG;

   // high nibble über BIT0..BIT3 senden
   DISPLAY_ENABLE_HIGH;
   LCD_DATA_OUT = data>>4;
   lcd_delay();
   DISPLAY_ENABLE_LOW;
   lcd_delay();


   // low nibble über BIT0..BIT3 senden
   DISPLAY_ENABLE_HIGH;
   LCD_DATA_OUT = data;
```

```
  lcd_delay();
  DISPLAY_ENABLE_LOW;
  lcd_delay();
}

//*****************************************************************************************************************
// Description  : Sends byte to the LCD's instruction-register in 8bit-mode
// Input        : data - data to send
// Output       : none
//*****************************************************************************************************************
void _SendByteToLCD_8BIT(unsigned char data)
{
// Daten sind gültig bei HIGH-LOW Flanke des enable-Signals !!
// for this function only control-registers are written
  SELECT_COMMAND_REG;

  DISPLAY_ENABLE_HIGH;
  LCD_DATA_OUT = data>>4;
  lcd_delay();
  DISPLAY_ENABLE_LOW;
  lcd_delay();
}


//*****************************************************************************************************************
// Description  : initialize the LCD controller
//              : needs about 30ms to start
// Input        : lcd_columns          - count of columns
// Output       : none
//*****************************************************************************************************************
void lcd_init(unsigned char lcd_columns)
{
// wait for more than 40ms after Vcc rises to 2.7V
// wait for more than 15ms after Vcc rises to 4.5V

  lcd_delay();lcd_delay();lcd_delay();lcd_delay();lcd_delay();lcd_delay();
  lcd_delay();lcd_delay();lcd_delay();lcd_delay();lcd_delay();lcd_delay();

  _lcd_maxx = lcd_columns;
  lcd_delay();
  LCD_DATA_DIR |= 0x000F;                                   // sets data lines as outputs
  LCD_COMM_DIR |= 0x0003;                                   // sets control lines as outputs

  DISPLAY_ENABLE_LOW;
//  DISPLAY_WRITE;

  lcd_delay();

  _SendByteToLCD_8BIT(0x30);                                // function set (Interface 8bit)
  lcd_delay();lcd_delay();

  _SendByteToLCD_8BIT(0x30);                                // function set (Interface 8bit)
  lcd_delay();lcd_delay();

  _SendByteToLCD_8BIT(0x30);                                // function set (Interface 8bit)
  lcd_delay();lcd_delay();

// initialisierungsdaten sind aus Datenblatt "DOTMATRIXDISPLAYS Electronic assembly" entnommen!
  _SendByteToLCD_8BIT(0x20);                                // function set 4bit (Interface 8bit)
  lcd_delay();
//  lcd_delay();
```

```
  _SendByteToLCD_4BIT(0x28,COMMAND);     // function set + number of display lines + character
font
  lcd_delay();
  _SendByteToLCD_4BIT(0x0F,COMMAND);                    // Display off
  lcd_delay();
  _SendByteToLCD_4BIT(0x01,COMMAND);                    // Display clear
  lcd_delay();
  _SendByteToLCD_4BIT(0x06,COMMAND);                    // entry mode set
  lcd_delay();
}

//*************************************************************************************************************************
// Description  : set the LCD display position x=0..39 y=0..3
//              : needs about 30ms to start
// Input        : x - X-position
//              : y - Y-position
// Output       : none
//*************************************************************************************************************************
void lcd_gotoxy(unsigned char x, unsigned char y)
{
  lcd_delay();
  _SendByteToLCD_4BIT(_base_y[y]+x, COMMAND);           // function set + 8bit bus
  _lcd_x=x;
  _lcd_y=y;
}

//*************************************************************************************************************************
// Description  : clears the LCD
// Input        : none
// Output       : none
//*************************************************************************************************************************
void lcd_clear(void)
{
  lcd_delay();
  _SendByteToLCD_4BIT(0x0c, COMMAND);                   // cursor off
  lcd_delay();
  _SendByteToLCD_4BIT(0x01, COMMAND);                   // function display clear

  _lcd_x=_lcd_y=0;

  lcd_delay();
}

//*************************************************************************************************************************
// Description  : write a char to the LCD
// Input        : c - character
// Output       : none
//*************************************************************************************************************************
void lcd_putchar(unsigned char c)
{

  if (_lcd_x>=_lcd_maxx)
  {
    _lcd_y++;
    lcd_gotoxy(0,_lcd_y);
  };

  lcd_delay();

  _SendByteToLCD_4BIT(c, DATA);          // send data
```

```
  _lcd_x++;
}

//*************************************************************************************************************************
// Description  : write the string str to the LCD
// Input        : *str            pointer to string
// Output       : none
//*************************************************************************************************************************
void lcd_puts(unsigned char *str)
{

  while(*str!=0x0)
  {
    lcd_putchar(*str);
    str++;
  }
}
```

1.2.7. Project_7_(UART_RS232)

In diesem Projekt wird die serielle Kommunikation über eine RS-232 Schnittstelle realisiert. Diese Schnittstelle wird über das UART-Modul des Mikrocontrollers. Durch diese Schnittstelle kann man mehrere MSP430 Education Systeme miteinander verbinden, Daten mit einem Computer oder einen anderen Gerät mit einer RS-232 Schnittstelle austauschen und externe Baugruppen ansteuern.
Die UART Funktionalität wird in den folgenden Beispielen mit dem Computer kontrolliert und aufgezeigt. Das UART Modul verwendet man mit dem Hyperterminal des Computers. Im Hyperterminal wird eine neue Verbindung mit dem angeschlossenen COM-Port hergestellt und mit den entsprechenden Parametern konfiguriert. Der einzige Parameter, welcher verändert werden muss, ist die Geschwindigkeit mit der eine Verbindung aufgebaut werden soll. Die richtige Geschwindigkeit wird immer im Namen der verwendeten Datei mitgeführt, so dass man diese im Hyperterminal einstellen kann.

1.2.7.1. uart01_02400_echo.c

MSP430F1232 Education System 1.3 - USART0 UART 2400 Ultra-low Power Echo ISR, 32 kHz ACLK

ACLK = UCLK0 = LFXT1 = 32768, MCLK = SMCLK = DCO ~ 800k
Baud rate divider with 32768hz XTAL @2400 = 32768Hz/2400 = 13.65 (000Dh)

Das Programm initialisiert das USART0 Modul als UART mit einer Bitgeschwindigkeit von 2400bit/s. Als Takt für den UART wird der ACLK verwendet, aus dem ACLK wird die Bitgeschwindigkeit durch entsprechendes Teilen und Modulieren gewonnen. Sobald der UART ein komplettes Byte empfangen hat, wird der RX Empfangs-Interrupt ausgeführt. Innerhalb der RX Interrupt Routine wird das empfangene Byte einfach in das TX Senderegister kopiert. Der UART generiert dann automatisch am TX-Pin das zugehörige Sendesignal. Die CPU befindet sich hauptsächlich im Low-Power-Mode, erst durch den Empfang eines Bytes und die damit verbundene Generierung des RX Interrupt wird die CPU aufgeweckt. Nach der Abarbeitung des RX ISR kehrt die CPU wieder in den Low-Power-Modus zurück.
Das Programm realisiert damit ein Echo. Das empfangene Byte wird an den Sender zurückgegeben.

```
#include <msp430x12x.h>

void main(void)
{
  WDTCTL = WDTPW + WDTHOLD;                 // Stop WDT
  UCTL0 = CHAR;                             // 8-bit character
  UTCTL0 = SSEL0;                           // UCLK = ACLK
  UBR00 = 0x0D;                             // 32k/2400 - 13.65
  UBR10 = 0x00;
  UMCTL0 = 0x6B;                            // Modulation
  ME2 |= UTXE0 + URXE0;                     // Enabled USART0 TXD/RXD
  IE2 |= URXIE0;                            // Enabled USART0 RX interrupt
  P3SEL |= 0x30;                            // P3.4,5 = USART0 TXD/RXD
  P3DIR |= 0x10;                            // P3.4 output direction
  _EINT();                                  // Enable interrupts

// Mainloop
  for (;;)
  {
  _BIS_SR(LPM3_bits);                       // Wait in LPM3 until character RXed
  while ((IFG2 & UTXIFG0) != UTXIFG0);      // USART0 TX buffer ready?
  TXBUF0 = RXBUF0;                          // RXBUF0 to TXBUF0
  }
}

// UART0 RX ISR will for exit from LPM3 in Mainloop
interrupt[UART0RX_VECTOR] void usart0_rx (void)
{
  _BIC_SR_IRQ(LPM3_bits);                   // Clear LPM3 bits from 0(SR)
}
```

1.2.8. MSP430_Edu_13_Test

Das MSP430_Edu_13_Test Projekt wurde zum Testen aller Funktionsgruppen entworfen. Es demonstriert die gemeinsame Verwendung mehrerer Funktionseinheiten in einem Programm und zeigt gleichzeitig die Funktion jeder einzelnen Baugruppe an.

1.2.8.1. MSP430_Edu_13_Test.c

MSP430F1232 Education System 1.3 - Ultimate Test Program

Dieses Programm dient zum Testen der komplett bestückten Entwicklungs-Platine. Den einzelnen Tastern wurden verschiedene Funktionen hinterlegt. Auf dem LCD wird jeweils angezeigt, welcher Taster betätigt wurde. Beim Programmstart wird das USART Modul als UART mit 2400bit/s initialisiert und sendet ein empfangenes Byte zurück an den Sender. Die LCD-Funktionalität wird ebenfalls durch die lcd.c realisiert wie sie in Kapitel 6 beschrieben wurde.

Funktionsbelegung der Taster:

- rot: Initialisierung der AD-Wandlung auf Kanal A6
- gelb: Initialisierung der AD-Wandlung auf Kanal A7
- grün: generiert einen Signalton
- blau: Schaltet UART Echo ein oder aus

```
#define   PWM_Offset   10
#define   ADConverter  BIT0
#define   TONE         BIT1
#define   RS232enable  BIT2
#include  <msp430x12x2.h>
#include  "lcd.c"                                          // include LCD driver

          // function prototypes
void SelectADC_Inch(unsigned char chn);
void DebounceDelay(void);
void DisableADC(void);
void SetupTA(unsigned char tone, unsigned char sec);
void StopTA(void);
void UARTtoggle(void);
void UARTenable(void);

          // variables
unsigned char string1[] = "Hallo! Ich bin";
unsigned char string2[] = "ein MSP430F1232";
unsigned char but1[] = "blauer Taster";
unsigned char but2[] = "gruener Taster";
unsigned char but3[] = "gelber Taster";
unsigned char but4[] = "roter Taster";
```

```
unsigned char tone_steps,sec_counter,StatusReg;

        // main function
void main(void)
{
  WDTCTL = WDTPW + WDTHOLD;                    // Stop watchdog timer
  UARTenable();                                // initialize UART0
  P2IE = BIT0+BIT1+BIT2+BIT5;                  // enable Interrupt
  P2DIR |= BIT4;                               // P2.4 as output
  lcd_init(16);                                // initialize LCD
  lcd_clear();                                 // clear all LCD digits
  lcd_puts(string1);                           // print string on LCD
  lcd_gotoxy(0,1);                             // move cursor
  lcd_puts(string2);                           // print string to LCD

  _EINT();                                     // global interrupt enable

  while(1)                                     // Repeat forever
  {
  }
}

        // interrupt hanlder for Port2 input pins
interrupt[PORT2_VECTOR] void P2_ISR(void) {
  DebounceDelay();                             // wait some time to debounce
  lcd_clear();                                 // clear all LCD digits
  if(P2IFG & BIT0) {                           // did P2.0 cause the isr?
    lcd_puts(but4);                            // yes, print string on LCD for button 4
    SelectADC_Inch(6);                  // initialize ADC10 to convert channel A6
    P2IFG &= ~ BIT0;                           // reset isr flag
  } else if(P2IFG & BIT1) {                    // did P2.1 cause the isr?
    lcd_puts(but3);                            // yes, print string on LCD for button 3
    SelectADC_Inch(7);                  // initialize ADC10 to convert channel A7
    P2IFG &= ~ BIT1;                           // reset isr flag
  } else if(P2IFG & BIT2) {                    // did P2.2 cause the isr?
    lcd_puts(but2);                            // yes, print string on LCD for button 2
    DisableADC();                       // disable ADC10, release Timer_A for other functions
    SetupTA(19,1);                      // setup Timer_A and output units to produce a beep
    P2IFG &= ~ BIT2;                           // reset isr flag
  } else if(P2IFG & BIT5) {                    // did P2.5 cause the isr?
    lcd_puts(but1);                            // yes, print string on LCD for button 1
    UARTtoggle();                              // select UART function
    P2IFG &= ~ BIT5;                           // reset isr flag
  }
}

        // UART functions
void UARTtoggle(void) {
  StatusReg ^= RS232enable;                    // toggle UART status bit
                                      // RS232enable = 1 -> UART Echo enabled
                                      // RS232enable = 1 -> UART Echo disenabled
}

void UARTenable(void) {
  StatusReg |= RS232enable;                    // enable UART Echo
  UCTL0 = CHAR;                                // 8-bit character
  UTCTL0 = SSEL0;                              // UCLK = ACLK
  UBR00 = 0x0D;                                // 32k/2400 - 13.65
  UBR10 = 0x00;                                // 2400bit/s
  UMCTL0 = 0x6B;                               // Modulation
  ME2 |= UTXE0 + URXE0;                        // Enabled USART0 TXD/RXD
```

```
  IE2 |= URXIE0;                                  // Enabled USART0 RX interrupt
  P3SEL |= 0x30;                                  // P3.4,5 = USART0 TXD/RXD
  P3DIR |= 0x10;                                  // P3.4 output direction
}

interrupt[UART0RX_VECTOR] void usart0_rx (void)
{
  if(StatusReg & RS232enable) {                   // Echo enabled?
    while ((IFG2 & UTXIFG0) != UTXIFG0);          // USART0 TX buffer ready?
    TXBUF0 = RXBUF0;                              // RXBUF0 to TXBUF0
  }
}

        // delay function to be used for debouncing the buttons by software
void DebounceDelay(void)
{
  unsigned int i;
  for (i = 50000; i > 0; i--);
}

void StopTA(void) {
  StatusReg &= ~TONE;                             // reset status bit
  P2SEL &= ~BIT4;                                 // select Port function for P2.4
  TACTL = 0;                                      // stop Timer_A
}
void SetupTA(unsigned char tone, unsigned char sec) {
  StatusReg |= TONE;                              // set status bit
  TACTL = TASSEL0+TACLR+TAIE;                     // ACLK, clear TA, overflow isr enable
  CCTL0 = OUTMOD_4 +CCIE;                         // CCR0 toggle mode
  CCTL2 = OUTMOD_4 +CCIE;                         // CCR2 toggle mode
  CCR0 = 32768;                                   // counter for seconds
  tone_steps = tone;                              // store parameter
  CCR2 = tone;                                    // init CCR2
  sec_counter = sec;                              // store parameter
  P2SEL |= BIT4;                                  // select modul function for P2.4
  TACTL |= MC1;                                   // Start Timer_a in continous mode
}

void DisableADC(void) {
  StatusReg &= ~ADConverter;                      // reset status bit
  ADC10CTL0 &= ~ENC;                              // ADC10 disabled
  ADC10CTL0 = 0;                                  // ADC10, Vref disabled completely
  TACTL = 0;                                      // stop Timer_A
  P2SEL &= ~BIT3;                                 // select Port function
  P2OUT &= ~BIT3;                                 // P2.3 as input
}

void SelectADC_Inch(unsigned char chn) {
  StatusReg |= ADConverter;
  switch(chn) {                                   // channel switch
    case 6:                                       // channel A6
      ADC10CTL1 = INCH_6+SHS_3;                   // P3.6, TA2 trigger sample start
      ADC10AE |= BIT6;                            // P3.6 ADC10 option select
      break;
    case 7:                                       // channel A7
      ADC10CTL1 = INCH_7+SHS_3;                   // P3.6, TA2 trigger sample start
      ADC10AE |= BIT7;                            // P3.6 ADC10 option select
      break;
  }
        // setup P1
  P1DIR = 0xFF;                                   // P1.0-P1.7 output
```

```
        // setup P2
  P2DIR |= BIT3;                                    // P2.3 as output
  P2SEL |= BIT3;                                    // select modul function TA2
        // setup CCR0
  CCTL0 = CCIE;                                     // Enable interrupt
  CCR0 = 0x3FF+PWM_Offset;                          // CCR0 duty cycle
        // setup CCR1
  CCTL1 = OUTMOD_7;                                 // CCR1 reset/set
  CCR1 = 1;                                         // CCR1 PWM Duty Cycle
        // setup CCR2
  CCTL2 = OUTMOD_3;                                 // CCR2 set/reset
  CCR2 = 2;                                         // CCR2 PWM Duty Cycle
        // setup Timer A
  TACTL = TASSEL0+MC0;                              // ACLK, up mode
}
interrupt[ADC10_VECTOR] void ADC10_ISR(void) {
  unsigned int adc_ref,bitmap;
  ADC10CTL0 &= ~ENC;                                // ADC10 disabled
  ADC10CTL0 = 0;                                    // ADC10, Vref disabled completely

  adc_ref = 0x7EF;
  bitmap = 0x1FF;
  while(bitmap > 0 && adc_ref > ADC10MEM) {
    bitmap = bitmap >> 1;
    adc_ref = adc_ref >> 1;
  }
  P1OUT = bitmap;
  CCR1 = ADC10MEM+PWM_Offset;
  if(!(StatusReg & RS232enable)) {
    while ((IFG2 & UTXIFG0) != UTXIFG0);            // USART0 TX buffer ready?
    TXBUF0 = 0xD;                   // send carriage return
    while ((IFG2 & UTXIFG0) != UTXIFG0);            // USART0 TX buffer ready?
    TXBUF0 = ADC10MEM;                              // send current ADC value
  }
}
interrupt[TIMERA0_VECTOR] void TA0_ISR(void) {
  if(StatusReg & TONE) {                            // if TONE status bit is set
    CCR0 += 32768;                                  // add offset
    sec_counter--;                                  // decrement counter
    if(sec_counter==0) {                            // sec_counter = 0?
      StopTA();                                     // stop Timer_A
      SelectADC_Inch(6);                    // initialize ADC10 to sample&convert A6
    }
  } else if(StatusReg & ADConverter) {              // if ADConverter status bit is set
    ADC10CTL0 = SREF_0+ADC10SHT_2+REFON+ADC10ON+ADC10IE;   // init ADC10, Vref on
    ADC10CTL0 |= ENC;                               // enable conversion
  }
}
        // Timer_A3 Interrupt Vector (TAIV) handler
interrupt [TIMERA1_VECTOR] void Timer_A1(void)
{
  switch( TAIV )
  {
  case  2: CCR1 += 0x3FFF;                          // Add Offset to CCR1
        break;
  case  4: CCR2 += tone_steps;                      // Add Offset to CCR2
        break;
  case 10: P1OUT ^= 0x01;                           // Timer_A3 overflow, toggle P1.0
        break;
 }
}
```

A. Quellen- und Literaturverzeichnis

[1] Texas Instruments
www.ti.com

[2] MSP430 Education System - Benutzerhandbuch