Mikroprozessorpraktikum WS 12/13

Carlos Martín Nieto, Simon Hohberg, Tu Tran

February 13, 2013

6 Timer

6.1 Zeitbasis

6.1.1 Finden Sie eine Lösung, die auf Basis eines Timer-Interrupts den Zustand der LED (P4.0) toggelt. Dazu muss der Timer jede Sekunde einen Interrupt generieren und in der ISR wird die LED getoggelt.

```
1  TBR = 0;
2
3  // waehle ACLK
4  BIT_SET(TBCTL, TBSSEL0);
5  BIT_CLR(TBCTL, TBSSEL1);
6
7  // wahle count up mode
8  BIT_SET(TBCTL, MC0);
9  BIT_CLR(TBCTL, MC1);
10
11 // setze taktanzahl fuer eine sekunde
12 TBCCR0 = 32769;
13
14 // loesche interrupt flag
15 BIT_CLR(TBCCTL0, CCIFG);
16
17 // interrupt freigeben
18 BIT_SET(TBCCTL0, CCIE);
19
20 LED_OFF(LED_ALL);
21
22  _bis_SR_register(GIE);
23
24 //===Hier die Endlosschleife quasi das Betriebssystem==================
25    while(1){
26
27      }    // Ende der Endlosschleife
28 }         // Ende Main
29 //====Ende des Hauptprogramms ========================================
30
31 #pragma vector = TIMERB0_VECTOR
32  __interrupt void TIMERB0(void)
33 {
34      // toggle die LED
35      LED_TOGGLE(LED_ROT);
36
37      BIT_CLR(TBCCTL0, CCIFG);
38 }
```

6.1.2 In welchen Low Powermodi würde der Sekundeninterrupt auf Basis der ACLK-Taktquelle funktionieren und in welchen nicht? Begründen Sie die Antwort.

In den Low Powermodi LPM0 - 3 würde dies funktionieren, da die ACLK laut Dokumentation in diesen Modi nicht abgeschaltet wird. Im LPM4 Modus würde dies hingegen nicht möglich sein, weil hier die ACLK abgeschaltet wird.

6.1.3 Nutzen Sie das Programm aus Aufgabe 6.1.1 als Ausgangsbasis. Setzen Sie den Divider des Timers unter Nutzung der IDx Bits auf 8. Erweitern Sie bitte das Programm in der Form, dass jeweils durch einen Tastendruck und eine Manipulation des TBCCR0 Registers:

- **Taster (P1.0), das Zeitintervall verdoppelt wird**
- **Taster (P1.1), das Zeitintervall halbiert wird**

Die Taster sollen im Interrupt genutzt werden und der zulässige Zahlenbereich von TBCCR0 darf nicht verlassen werden.

```
1  TBR = 0;
2
3  // waehle ACLK
4  BIT_SET(TBCTL, TBSSEL0);
5  BIT_CLR(TBCTL, TBSSEL1);
6
7  // wahle count up mode
8  BIT_SET(TBCTL, MC0);
9  BIT_CLR(TBCTL, MC1);
10
11 // setze taktanzahl fuer eine sekunde
12 TBCCR0 = 32769;
13
14 // loesche interrupt flag fuer timer
15 BIT_CLR(TBCCTL0, CCIFG);
16
17 BIT_CLR(P1IES, TASTE_LINKS | TASTE_RECHTS); // LH an linker taste loest interrupt aus
18 BIT_CLR(P1IFG, TASTE_LINKS | TASTE_RECHTS); // interrupt flag loeschen
19 BIT_SET(P1IE, TASTE_LINKS | TASTE_RECHTS); // interrupts fuer linke taste erlauben
20
21 // interrupt fuer timer freigeben
22 BIT_SET(TBCCTL0, CCIE);
23
24 LED_OFF(LED_ALL);
25
26  _bis_SR_register(GIE);
27
28 //===Hier die Endlosschleife quasi das Betriebssystem==================
29    while(1){
30
31      }    // Ende der Endlosschleife
32 }         // Ende Main
33 //====Ende des Hauptprogramms ========================================
34
35 #pragma vector = TIMERB0_VECTOR
36  __interrupt void TIMERB0(void)
37 {
38      LED_TOGGLE(LED_ROT);
39
40 }
41
42 #pragma vector = PORT1_VECTOR
```

```
43 __interrupt void PORT1(void)
44 {
45     unsigned long new_value = 0;
46
47     // wenn linker taster gedrueckt wird
48     if (P1IFG & TASTE_LINKS) {
49         BIT_CLR(P1IFG, TASTE_LINKS);
50
51         // verdoppele den aktuellen wert
52         new_value = (long)TBCCR0 * (long)2;
53
54         // ueberpruefe, ob der neue wert die obere grenze ueberschreitet
55         if (new_value > 0xffff) {
56             new_value = 0xffff;
57         }
58     }
59
60     // wenn rechter taster gedrueckt wird
61     if (P1IFG & TASTE_RECHTS) {
62         BIT_CLR(P1IFG, TASTE_RECHTS);
63
64         // halbiere den aktuellen wert
65         new_value = TBCCR0 / 2;
66
67         // ueberpruefe untere grenze
68         if (0 == new_value) {
69             new_value = 1;
70         }
71     }
72
73     // setze counter zurueck
74     TBR = 0;
75     // setze neuen zielwert
76     TBCCR0 = new_value;
77 }
```

6.2 Zeitmessung

6.2.1 Entwickeln Sie ein Programm, das die Zeit zwischen zwei Tastenbetätigungen (P1.0 und P1.1) auf Basis des Timers B misst.

```
1  static unsigned int cycles = 0;
2
3  .
4  .
5  .
6
7  // interrupt wird bei 0 ausgeloest
8  TBR = 1;
9
10 // waehle ACLK
11 BIT_SET(TBCTL, TBSSEL0);
12 BIT_CLR(TBCTL, TBSSEL1);
13
14 // setze divider auf 8 -> 4096 Hz
15 BIT_SET(TBCTL, ID0 | ID1);
16
17 // wahle count up mode
18 BIT_SET(TBCTL, MC0);
```

```
19 BIT_CLR(TBCTL, MC1);
20
21 // setze taktanzahl fuer eine sekunde
22 TBCCR0 = 0xffff;
23
24 // loesche interrupt flag fuer timer
25 BIT_CLR(TBCCTL0, CCIFG);
26
27 BIT_CLR(P1IES, TASTE_LINKS | TASTE_RECHTS); // LH an linker taste loest interrupt aus
28 BIT_CLR(P1IFG, TASTE_LINKS | TASTE_RECHTS); // interrupt flag loeschen
29 BIT_SET(P1IE, TASTE_LINKS | TASTE_RECHTS); // interrupts fuer linke taste erlauben
30
31 LED_OFF(LED_ALL);
32
33 _bis_SR_register(GIE);
34
35 //===Hier die Endlosschleife quasi das Betriebssystem==================
36   while(1){
37
38     }   // Ende der Endlosschleife
39 }       // Ende Main
40 //====Ende des Hauptprogramms ========================================
41
42 #pragma vector = TIMERB0_VECTOR
43 __interrupt void TIMERB0(void)
44 {
45     cycles++;
46 }
47
48 #pragma vector = PORT1_VECTOR
49 __interrupt void PORT1(void)
50 {
51     unsigned int tbr = TBR;
52     unsigned int secs = 0;
53     unsigned int centi_secs = 0;
54     char buffer[16];
55
56     // linker taster
57     if (P1IFG & TASTE_LINKS) {
58
59         // starte zeitmessung
60         BIT_CLR(P1IFG, TASTE_LINKS);
61
62         cycles = 0;
63         TBR = 1;     // interrupt wird ausgeloest bei 0
64
65         // interrupt fuer timer freigeben
66         BIT_SET(TBCCTL0, CCIE);
67     }
68
69     // rechter taster
70     if (P1IFG & TASTE_RECHTS) {
71
72         // stoppe zeitmessung
73         BIT_CLR(P1IFG, TASTE_RECHTS);
74
75         // interrupt fuer timer stoppen
76         BIT_CLR(TBCCTL0, CCIE);
77
78         secs = cycles * 16 + tbr / 4096;
79         centi_secs = (tbr % 4096) % 10;
80
```

```
81          // ausgabe
82          sprintf(buffer, "Zeit: %02d,%02d", secs, centi_secs);
83          lcd_clear(WHITE);
84          lcd_string(BLACK, 15, 25, buffer);  // Textausgabe
85          lcd_paint();
86      }
87  }
```

6.3 Zeitschalterk

6.3.1 Programmieren Sie eine Uhr, die ein Format der Form hh:mm:ss realisiert und integrieren Sie eine Schaltfunktion für eine LED P4.2 (kleinste Zeiteinheit ist die Sekunde). Zwei Variablen (t1 und t2) dienen zum speichern der Schaltzeiten.

```
1   static unsigned long t1 = 60*60*8 + 60*30 + 22;
2   static unsigned long t2 = 60*60*9 + 60*1 + 01;
3   static unsigned char hours = 0;
4   static unsigned char mins = 0;
5   static unsigned char secs = 0;
6
7   .
8   .
9   .
10
11  // interrupt wird bei 0 ausgeloest
12  TBR = 1;
13
14  // waehle ACLK
15  BIT_SET(TBCTL, TBSSEL0);
16  BIT_CLR(TBCTL, TBSSEL1);
17
18  // setze divider auf 8
19  BIT_SET(TBCTL, ID0 | ID1);
20
21  // wahle count up mode
22  BIT_SET(TBCTL, MC0);
23  BIT_CLR(TBCTL, MC1);
24
25  // setze taktanzahl fuer eine sekunde
26  TBCCR0 = 4096; // eine sekunde: takt=32000 / divider=8
27
28  // loesche interrupt flag fuer timer
29  BIT_CLR(TBCCTL0, CCIFG);
30
31  BIT_CLR(P1IES, TASTE_LINKS | TASTE_RECHTS); // LH an linker taste loest interrupt aus
32  BIT_CLR(P1IFG, TASTE_LINKS | TASTE_RECHTS); // interrupt flag loeschen
33  BIT_SET(P1IE, TASTE_LINKS | TASTE_RECHTS); // interrupts fuer linke taste erlauben
34
35  LED_OFF(LED_ALL);
36
37  BIT_SET(TBCCTL0, CCIE);
38
39  _bis_SR_register(GIE);
40
41  //===Hier die Endlosschleife quasi das Betriebssystem==================
42    while(1){
43
44      }   // Ende der Endlosschleife
45  }       // Ende Main
```

```
46 //====Ende des Hauptprogramms ========================================
47
48 #pragma vector = TIMERB0_VECTOR
49 __interrupt void TIMERB0(void)
50 {
51     char buffer[16];
52     secs++;
53
54     // erhoehe sekundenzaehler und beruecksichtige ueberlauf
55     if (secs > 59) {
56         mins++;
57         secs %= 60;
58         if (mins > 59) {
59             mins %= 60;
60             hours++;
61             if (hours > 23) {
62                 hours %= 24;
63             }
64         }
65     }
66
67     // ueberpruefe schaltzeit 1
68     if (t1 == 60*60*(long)hours + 60*(long)mins + (long)secs) {
69         LED_ON(LED_GELB);
70     }
71
72     // ueberpruefe schaltzeit 2
73     if (t2 == 60*60*(long)hours + 60*(long)mins + (long)secs) {
74         LED_OFF(LED_GELB);
75     }
76
77     // ausgabe
78     sprintf(buffer, "%02d:%02d:%02d", hours, mins, secs);
79     lcd_clear(WHITE);
80     lcd_string(BLACK, 15, 25, buffer);  // Textausgabe
81     lcd_paint();
82     LED_TOGGLE(LED_ROT);
83 }
```

6.4 LED als Indikator

6.4.1 6.4.1 Soll eine LED (P4.0) nur die Bereitschaft eines Controllers signalisieren, ist es energetisch sinnvoll alle 5 Sekunden für die Zeit von 0,5 Sekunden die LED einzuschalten. Beispiele dafür sind Alarmanlagen, BT-GPS-Empfänger und Handys.
Der Controller befindet sich bei dieser Aufgabe in einer Endlosschleife im LPM3-Mode und wird durch den Timerinterrupt "aufgeweckt".
Programmieren Sie den Timer B mit einem CCR0-Interrupt in der Form, dass per ISR die LED im oben beschriebenen Rhythmus blinkt.

6.4.2 Finden Sie dabei eine Lösung, die es ihnen gestattet den Rhythmus relativ einfach zu verändern. Testen Sie das Programm mit dem Zyklus 3 Sekunden aus und 1 Sekunde an.

```
1 #define IS_LED_ON(led) (!(P4OUT & led))
2 #define TIMER_INTERRUPT_MS 3000
3 #define LED_ON_MS 1000
4
5 // interrupt wird bei 0 ausgeloest
6 TBR = 1;
```

```
7
8  // waehle ACLK
9  BIT_SET(TBCTL, TBSSEL0);
10 BIT_CLR(TBCTL, TBSSEL1);
11
12 // setze divider auf 8
13 BIT_SET(TBCTL, ID0 | ID1);
14
15 // wahle count up mode
16 BIT_SET(TBCTL, MC0);
17 BIT_CLR(TBCTL, MC1);
18
19 // setze anzahl fuer interrupt
20 TBCCR0 = (4 * TIMER_INTERRUPT_MS); // eine sekunde: takt=32000 / divider=8
21
22 // loesche interrupt flag fuer timer
23 BIT_CLR(TBCCTL0, CCIFG);
24
25 LED_OFF(LED_ALL);
26
27 BIT_SET(TBCCTL0, CCIE);
28
29 _bis_SR_register(GIE);
30
31 //===Hier die Endlosschleife quasi das Betriebssystem==================
32     while(1){
33         LPM3;
34     }    // Ende der Endlosschleife
35 }        // Ende Main
36 //====Ende des Hauptprogramms ========================================
37
38 #pragma vector = TIMERB0_VECTOR
39 __interrupt void TIMERB0(void)
40 {
41     if (IS_LED_ON(LED_ROT)) {
42         // setze counter fuer die zeit in der die LED aus sein soll
43         TBCCR0 = (4 * TIMER_INTERRUPT_MS);
44         LED_OFF(LED_ROT);
45     } else {
46         // setze counter fuer die zeit in der die LED an sein soll
47         TBCCR0 = (4 * LED_ON_MS);
48         LED_ON(LED_ROT);
49     }
50 }
```

6.4.3 Welche Batterienutzungsdauer wird für beide Blinkmodi erreicht, wenn eine 1100mAh Batterie genutzt wird. Es wird dabei vorausgesetzt, dass der Gesamtstromverbrauch des Controllerboards bei 5mA (LED an) und bei 100µA (LED aus) liegt. Zu welchem Ergebnis kommen Sie?

- Durchschnittlicher Verbrauch

  $\frac{3s \cdot 100\,\mu A + 1s \cdot 5\,mA}{4s} = \frac{5{,}3\,mAs}{4s} = \underline{1{,}325\,mA}$

- Batterienutzungsdauer

  $\frac{1100\,mAh}{1{,}325\,mA} \approx \underline{830{,}189\,h \approx 34{,}6\,d}$

6.5 PWM

6.5.1 Machen Sie sich mit der Funktionsweise und der technischen Anwendung der Pulsweitenmodulation vertraut. Erklären Sie kurz die Funktionsweise der PWM-Ansteuerung für folgende technische Anwendungen:

- Motorsteuerung

  Ein Motor lässt sich mit Hilfe von PWM insofern ansteuern, dass die mittlere Spannung an dem Motor die Geschwindigkeit des Motors bestimmt. D.h. immer wenn das PWM Signal gesetzt ist, beschleunigt der Motor und wenn es nicht gesetzt ist, wird der Motor durch Reibung wieder langsamer. Je größer der Anteil der Zeit in der das PWM Signal gesetzt ist, desto schneller dreht sich der Motor.

- Ansteuerung bzw. Dimmen von LED

  Beim Dimmen verhält es sich ähnlich. Wenn Das PWM Signal gesetzt ist, fließt Strom, der die LED leuchten lässt. Wenn das Signal nicht gesetzt ist, ist auch die LED aus. D.h. die LED flackert. Ist nun die Frequenz dieses Flackerns groß genug, kann das menschliche Auge das An- und Ausschalten der LED nicht mehr wahrnehmen und man sieht nur noch die "durchschnittliche" Helligkeit der LED. Die LED wirkt also immer dunkler, je länger die LED durchschnittlich von dem PWM Signal ausgeschaltet wird.

6.5.2 Machen Sie sich mit dem Kapitel 12 aus dem MSP430 User's Guide vertraut. Schauen Sie sich insbesondere den Punkt 12.2.5 im MSP430 User's Guide an. Wählen Sie einen Ansteuerungsmode.

Wir wählen den count-up Modus, so dass der Timer wiederholt bis zum Wert in dem Register TBCL0 zählt. Als out-mode der Output-Unit wählen wir den Set/Reset Modus, was dazu führt, dass das Ausgangssignal gesetzt wird, wenn der Timer den Wert in TBCLx erreicht hat und zurückgesetzt wird, wenn der Timer den Wert in TBCL0 erreicht hat. Dies ermöglicht uns sowohl die Dauer in der das Ausgangssignal gesetzt ist, als auch die Dauer in der das Ausgangssignal nicht gesetzt ist, variabel festzulegen.

6.5.3 Nutzen Sie den Timer B und eine LED an der Portleitung P4.1, um einen PWM Generator softwaretechnisch entsprechend A 6.5.2 umzusetzen. Denken Sie daran, die entsprechenden P4SEL Bits zu Setzen. Das System soll ohne Interrupts betrieben werden.
In welchen Zeitbereichen für die Periodendauer T läßt sich eine PWM auf Basis der ACLK-Taktquelle realisieren? Die Einstellungen können Sie einfach im Debug Mode der Entwicklungsumgebung vornehmen. Die Zeiten können Sie mit den vorhandenen Messgeräten (Counter bzw. Oszi) bestimmen.
Dimmen Sie die LED. Die Periodendauer T sollte bei einer Frequenz oberhalb von 25 Hz liegen, ansonsten wird das Ein- und Ausschalten der LED visuell wahrnehmbar.

```
1  BIT_SET(P4SEL, BIT1);
2
3  // waehle ACLK
4  BIT_SET(TBCTL, TBSSEL0);
5  BIT_CLR(TBCTL, TBSSEL1);
6
7  // setze divider auf 8
8  //BIT_SET(TBCTL, ID0 | ID1);
9
10 // setze out-mode 011, set/reset mode
11 BIT_SET(TBCCTL1, OUTMOD0);
12 BIT_SET(TBCCTL1, OUTMOD1);
13 BIT_CLR(TBCCTL1, OUTMOD2);
14
```

```
15 // wahle count up mode
16 BIT_SET(TBCTL, MC0);
17 BIT_CLR(TBCTL, MC1);
18
19 TBCCR0 = 512;
20 TBCCR1 = 20;
21
22 LPM3;
23 //===Hier die Endlosschleife quasi das Betriebssystem==================
24   while(1){
25     }   // Ende der Endlosschleife
```

- Zeitbereiche für die Periodendauer

  Maximalfrequenz der ACLK: $32000\,Hz$
  Minimalfrequenz der ACLK: $4000\,Hz$

  Daraus ergeben sich die Periodendauern:

  - $T_{min} = \frac{1}{32000\,Hz} = 3,125 \cdot 10^{-5}\,s = \underline{31,25\,\mu s}$
  - Wenn TBCCR1 gleich dem größten Wert der für TBCCR0 möglich ist, ergibt sich die maximale Periodendauer:
    $T_{max} = \frac{2^{16}-1}{4000\,Hz} = \underline{16,3835\,s}$