Mikroprozessorpraktikum WS 12/13

Carlos Martín Nieto, Simon Hohberg, Tu Tran

January 31, 2013

4 Interrupt

4.1 Taster

4.1.1 Bisher konnten wir den Zustand einer Portleitung nur durch fortwährendes Abfragen testen. Dahinter steht der Begriff Polling. Nun wollen wir die Interruptfähigkeit des Ports P1 nutzen. Der Effekt besteht darin, daß hardwaremäßig eine LH- bzw. HL-Flanke automatisch eine Interrupt Service Routine (ISR) startet.

- Teilaufgabe 1:

```
1  // konfiguriere Taster-Interrupts
2  BIT_CLR(P1IES, TASTE_LINKS); // LH
3  BIT_CLR(P1IES, TASTE_RECHTS); // LH
4
5  // erlaube Interrupt
6  BIT_SET(P1IE, TASTE_LINKS | TASTE_RECHTS);
7
8  // loesche Interruptflags
9  BIT_CLR(P1IFG, TASTE_LINKS | TASTE_RECHTS);
10
11
12 _bis_SR_register(GIE);
13
14 LED_OFF(LED_ALL);
15
16 //===Hier die Endlosschleife quasi das Betriebssystem==================
17 while(1){
18     }   // Ende der Endlosschleife
19 }       // Ende Main
20 //====Ende des Hauptprogramms ========================================
21
22 #pragma vector = PORT1_VECTOR
23 __interrupt void PORT1(void) {
24
25     // wenn Interrupt fuer linken Taster
26     if (P1IFG & TASTE_LINKS) {
27         BIT_CLR(P1IFG, TASTE_LINKS);
28         LED_TOGGLE(LED_GRUEN);
29     }
30
31     // wenn Interrupt fuer rechten Taster
32     if (P1IFG & TASTE_RECHTS) {
33         BIT_CLR(P1IFG, TASTE_RECHTS);
34         LED_TOGGLE(LED_ROT);
```

```
35      }
36  }
```

- Teilaufgabe 2:

```
1   // konfiguriere Taster-Interrupts
2   BIT_CLR(P1IES, TASTE_LINKS); // LH
3   BIT_SET(P1IES, TASTE_RECHTS); // HL
4
5   // erlaube Interrupt
6   BIT_SET(P1IE, TASTE_LINKS | TASTE_RECHTS);
7
8   // loesche Interruptflags
9   BIT_CLR(P1IFG, TASTE_LINKS | TASTE_RECHTS);
10
11
12  _bis_SR_register(GIE);
13
14  LED_OFF(LED_ALL);
15
16  //===Hier die Endlosschleife quasi das Betriebssystem==================
17  while(1){
18      }   // Ende der Endlosschleife
19  }       // Ende Main
20  //====Ende des Hauptprogramms ========================================
21
22  #pragma vector = PORT1_VECTOR
23  __interrupt void PORT1(void) {
24
25      // wenn Interrupt fuer linken Taster
26      if (P1IFG & TASTE_LINKS) {
27          BIT_CLR(P1IFG, TASTE_LINKS);
28          LED_TOGGLE(LED_GRUEN);
29      }
30
31      // wenn Interrupt fuer rechten Taster
32      if (P1IFG & TASTE_RECHTS) {
33          BIT_CLR(P1IFG, TASTE_RECHTS);
34          LED_TOGGLE(LED_ROT);
35      }
36  }
```

- Teilaufgabe 3:

```
1   // konfiguriere Taster-Interrupts
2   BIT_CLR(P1IES, TASTE_LINKS); // LH
3   BIT_SET(P1IES, TASTE_RECHTS); // HL
4
5   // erlaube Interrupt
6   BIT_SET(P1IE, TASTE_LINKS | TASTE_RECHTS);
7
8   // loesche Interruptflags
9   BIT_CLR(P1IFG, TASTE_LINKS | TASTE_RECHTS);
10
11
12  _bis_SR_register(GIE);
13
14  LED_OFF(LED_ALL);
```

```
15
16 //===Hier die Endlosschleife quasi das Betriebssystem==================
17 while(1){
18     }   // Ende der Endlosschleife
19 }       // Ende Main
20 //====Ende des Hauptprogramms ========================================
21
22 __interrupt void PORT1(void) {
23     static int count = 0;
24
25     // wenn Interrupt fuer linken Taster
26     if (P1IFG & TASTE_LINKS) {
27         BIT_CLR(P1IFG, TASTE_LINKS);
28         LED_TOGGLE(LED_GRUEN);
29         count++;
30         if (count == 10) {
31             LED_ON(LED_GELB);
32             // Als Input setzen, sodass keine Interrupts moeglich sind
33             BIT_SET(P1SEL, TASTE_LINKS);
34         }
35     }
36
37     // wenn Interrupt fuer rechten Taster
38     if (P1IFG & TASTE_RECHTS) {
39         BIT_CLR(P1IFG, TASTE_RECHTS);
40         LED_TOGGLE(LED_ROT);
41     }
42 }
```

4.2 Totmann

4.2.1 Der Watchdog und ein Tasten-Interrupt sollen in dieser Aufgabe gleichzeitig genutzt werden. In der Technik kommt so etwas in Form einer sogenannten Totmannschaltung vor. Betätigt der Fahrer eines Fahrzeugs innerhalb eines festgelegten Zeitraumes nicht eine Taste, erfolgt eine Notbremsung.

```
1  // konfiguriere Interrupts fuer Taster
2  BIT_CLR(P1IES, TASTE_LINKS); // LH
3  BIT_SET(P1IE, TASTE_LINKS);  // erlaube Interrupts
4  BIT_CLR(P1IFG, TASTE_LINKS); // loesche Interruptflag
5
6  LED_OFF(LED_ALL);
7
8  // setze ACLK divider auf 4
9  BIT_SET(BCSCTL1, DIVA1);
10
11 // konfiguriere Watchdog:
12 // Passwort, Counter zuruecksetzen, ACKL, Interval Timer Mode
13 WDTCTL = WDTPW + WDTCNTCL + WDTSSEL + WDTTMSEL;
14
15 // erlaube Interrupt fuer Watchdog
16 BIT_SET(IE1, WDTIE);
17
18 LED_ON(LED_GELB);
19
20 _bis_SR_register(GIE);
21
22 //===Hier die Endlosschleife quasi das Betriebssystem==================
23     while(1){
```

```
24          LED_TOGGLE(LED_GRUEN);
25          wait(50000);
26      }   // Ende der Endlosschleife
27  }       // Ende Main
28  //====Ende des Hauptprogramms ========================================
29
30  #pragma vector = PORT1_VECTOR
31  __interrupt void PORT1(void)
32  {
33      // wenn linker Taster gedrueckt
34      if (P1IFG & TASTE_LINKS) {
35              BIT_CLR(P1IFG, TASTE_LINKS);
36              // Watchdog neu knofigurieren
37              WDTCTL = WDTPW + WDTCNTCL + WDTSSEL + WDTTMSEL;
38              LED_OFF(LED_GELB);
39      }
40  }
41
42  #pragma vector = WDT_VECTOR
43  __interrupt void WDT(void)
44  {
45      // Interruptflag zuruecksetzen
46      BIT_CLR(IFG1, WDTIFG);
47
48      // wenn gelbe LED aus ist, schalte sie wieder an
49      if (P4OUT & LED_GELB) {
50          LED_ON(LED_GELB);
51      } else {
52          // Watchdog stoppen
53          WDTCTL = WDTPW + WDTHOLD;
54
55          // fuehre Ampelsequenz aus
56          while (true) {
57              LED_OFF(LED_GELB | LED_GRUEN | LED_ROT);
58              LED_ON(LED_ROT);
59              SLEEP(1);
60              LED_ON(LED_GELB);
61              SLEEP(1);
62              LED_OFF(LED_GELB | LED_ROT);
63              LED_ON(LED_GRUEN);
64              SLEEP(1);
65              LED_OFF(LED_GRUEN);
66              LED_ON(LED_GELB);
67              SLEEP(1);
68          }
69      }
70  }
```

4.3 Touchscreen

4.3.1 In der Aufgabe A 1.5.1 ist der Touchscreen schon als Taste genutzt worden. Allerdings wurde die Funktion unter Nutzung von Polling realisiert. In dieser Aufgabe soll die gleiche Aufgabenstellung unter Nutzung des Touchscreens als Interruptquelle gelöst werden. Bitte beachten Sie, daß nur der Touchscreen einen Interrupt auslösen kann. Die beiden Taster werden in der Interruptservice nur eingelesen um die Zählrichtung festzustellen. Stellen Sie die Anzahl der Touchscreen (TS) Berührungen auf dem LCD-Display dar und aktualisieren Sie diese ständig.

```
// alle Leitungen auf Eingang
    TS_TIP_DIR_IN;
    TS_YP_DIR_IN;
    TS_YM_DIR_IN;
    TS_XP_DIR_IN;
    TS_XM_DIR_IN;
    // die Ausgangsregister vorbereitend setzen
    TS_XM_0; // XM X-Achse wird auf 0 gesetzt
    TS_TIP_1; // YP Y-Achse wird über einen PullUp Widerstand auf 1 gezogen
    // Die Ausgaenge jetzt freigeben
    TS_XM_DIR_OUT;  // XM auf 0
    TS_TIP_DIR_OUT; // YP auf 1
    BIT_SET(P1IE, BIT6);
    BIT_CLR(P1IFG, BIT6);
     _bis_SR_register(GIE);
//===Hier die Endlosschleife quasi das Betriebssystem==================
print_value();
while(1){
    // linker Taster gedrueckt -> gruen an
    if (P1IN & TASTE_LINKS) {
        LED_OFF(LED_ROT | LED_GELB);
        LED_ON(LED_GRUEN);
    // rechter Taster gedrueckt -> rot an
    } else if (P1IN & TASTE_RECHTS) {
        LED_OFF(LED_GRUEN | LED_GELB);
        LED_ON(LED_ROT);
    // kein Taster gedrueckt -> gelb an
    } else {
        LED_OFF(LED_GRUEN | LED_ROT);
        LED_ON(LED_GELB);
    }
}   // Ende der Endlosschleife
}   // Ende Main //====Ende des Hauptprogramms ========================================
void print_value(void)
{
    char string[256];
    sprintf(string, "DEC %03d\nHEX %03x", wert, wert);
    lcd_clear(WHITE);
    // Grafikspeicher auf dem MSP430 löschen
    lcd_string(BLACK, 15, 25, string);
    // Textausgabe  lcd_paint();
    // Grafikspeicher auf das LCD Display ausgeben
}
#pragma vector = PORT1_VECTOR __interrupt void PORT1(void)
{
    if (P1IFG & BIT6) {
        // linker Taster gedrueckt -> wert erniedrigen
        if (P1IN & TASTE_LINKS) {
            wert--;
        // rechter Taster gedrueckt -> wert erhoen
        } else if (P1IN & TASTE_RECHTS) {
            wert++;
        // keiner gedrueckt -> gelb aus, solange gedrueckt ist
        } else {
            LED_TOGGLE(LED_GELB);
            while (!(P1IN & BIT_YM)) {
            }
        }
        print_value();
        BIT_CLR(P1IFG, BIT6);
    }
}
```