Mikroprozessor Praktikum
Fr. HWP 2
Aufgabenblock 4

Paul Podlech
3910583

Max Wisniewski
4370074

6. März 2012

Block 4: Interrupt

Aufgabe 4.1 : Taster

A 4.1.1

Wir wollen die Interruptfähigkeit des Ports 1 nutzen. Der Effekt besteht darin, dass nicht ständig ein Polling des Zustands einer Portleitung erforderlich ist, um den Zustand der Taster zu erkennen.

Unsere Beispielanwendung besteht im Kern aus einer Endlosschleife, die eine wait() Funktion und eine Codezeile zum toggeln der LED (P4.2) beinhaltet.

Beide Taster an Port1 (P1.0 und P1.1) sollen interruptfähig sein und auf eine LH-Flanke den Interrupt auslösen. Die notwendige Initialisierung der Register des Ports P1 und die Freigabe des Interrupts muß vor der Endlosschleife erfolgen.

In der ISR Interrupt-Service-Routine für den Port1 soll folgende Funktionalität integriert werden:

- für den Taster an P1.0: nach jedem 10-ten Tastendruck soll die LED (P4.0) getoggelt werden
- für den Taster an P1.1: bei jedem Tastendruck soll die LED (P4.1) getoggelt werden

Im nächsten Schritt soll eine Lösung gefunden werden, die eine mögliche Interruptquelle ausschalten kann.

Verändern Sie dazu die ISR in der Form:

- für den Taster an P1.0:
  nach jedem 10-ten Tastendruck soll die LED (P4.0) getoggelt werden
  nur bei leuchtender LED (P4.1) soll der Taster (P1.0) interruptfähig sein

- für den Taster an P1.1:
  bei jedem Tastendruck soll die LED (P4.1) getoggelt werden

  Testen und dokumentieren Sie ihre Lösung.

Programm:

```
1  #pragma vector = PORT2_VECTOR
2  __interrupt void PORT1 (void) {
3    if(P1IN & 0x01){
4      dosmth1();
5    }
6    if(P1IN & 0x02){
7      dosmth2();
8    }
9    P1IFG &= ~(0xFF);
10 }
11
12 char counter1 = 0;
13 char counter2 = 0;
14
15 void dosmth1(){
16   P1IE &= ~(0x01);
17   if(counter1%20 == 9){
18     LED1ON;
19   }
20   if(counter1%20 == 19){
21     LED1OFF;
22   }
23   counter1 = (counter1 + 1) %20;
```

```
24    wait(10000);
25    P1IE |= 0x01;
26  }
27
28  void dosmth2(){
29    P1IE &= ~(0x02);
30    if(counter2 == 1){
31      P1IE &= ~(0x01);
32      LED2OFF;
33      counter2 = 0;
34    }
35    else{
36      P1IE |= 0x01;
37      LED2ON;
38      counter2 = 1;
39    }
40    wait(10000);
41    P1IC |= 0x02;
42  }
```

Erklärung:

In Zeile 1-2 haben wir eine neue ISR angelegt, die über den Compiler für den zweiten Port in die Interruptvektortabelle gelegt wird. Innerhalb dieser Testen wir zunächst, welcher Taster nun wirklich gedrückt wurde. Danach entscheiden wir, wie wir weiter verfahren. Am ende setzen wir die Interruptfalg auf 0 zurück, womit der Interrupt als erledigt gesetzt wird.

Wurde Taster 1 gedrückt, deaktivieren wir für diesen Taster zunächst die Interrupts. Ist der Counter nun im 10. Schritt, schalten wir die LED1 an und sind wir im 20. Schritt ist die LED1 aus. Danach aktivieren wir die Interrupts für diese Taste wieder.

Haben wir die zweite Taste gedrückt, testen wir, ob unsere LED an sein müsste. Ist sie an, deaktivieren wir den Interrupt für die erste Taste (zweiter Aufgabenteil, will man dies nicht, so muss man nur Zeile 31 und 36 auskommentieren) schalten die LED aus. Ist die LED aus, aktivieren wir sie und schalten den Interrupt für den Taster 1 weider an.

Beobachtung:

Das Programm verhaltet sich wie gewünscht. Wir haben uns diesmal allerdings nicht weiter um das Prellen gekümmert. Dadurch gestaltete sich das Testen manchmal schwierig. Der Taster 1 funktionierte aber nur, wenn die LED 2 an ist.

Aufgabe 4.2 : Totmann

A 4.1.1

In dieser Aufgabe soll eine Totmanschaltung realisiert werden. Wird ein Taster innerhalb eines gewissen Zeitraumes nicht gedrückt, so erfolgt eine spezifizierte Reaktion darauf. Zum Beispiel eine Notbremsung, Explosion von Sprengstoff und viele spaßige Dinge mehr.

Dabei soll das folgende Passieren:

1. Taste (P1.0) als Interruptquelle programmieren
2. alle LED ausschalten
3. Wathcdog als Timer Interruptquelle programmieren
4. LED (P4.1) einschalten
5. Endlosschleife mit 0.5 Sekundentakt blinkender LED (P4.0)

6. Bei Tastendruck (P1.0) Watchdogteimer neu Starten und (P4.1) auschalten.

Springt der Watchdog an, soll das folgende Passieren:

- Ist LED (P4.1) aus, wird sie wieder eingeschaltet
- Ist LED (P4.1) an, wird der Watchdog Timer Interrupt ausgeschaltet und eine Ampelsequenz dauerhaft angeschaltet.

Programm:

```
1
2  //Entscheidet in welcher Phase wir sind
3  char flag42 = 0;
4
5  void aufgabe42init(){
6    // Uebliche Initialisierung
7    P4SEL &= ~(0x07);
8    P4DIR |= 0x07;
9    P4OUT |= 0x07;
10
11   BSCTL! |= DIVA_2; // Basetakt ist 1s, DIVA_2 bedeute x4
12   // Startet den Watchdog, mit dem TimerMode
13   WDTCTL = DWTPW + WDTCNTCL + WDTSSEL + WDTTMSEL;
14   IE1 |= WDT_IE;
15   IE1 |= 0x01;
16   P1IES &= ~(0x01);
17 }
18
19 #pragma vector = WDT_VECTOR
20 __interrupt void do_wdt(void){
21   if(P4IN & 0x02){
22     LED2ON;
23     WDTCTL = WDTPW + WDTCNTCL + WDTSSEL + WDTTMSEL;
24   } else {
25     WDTCTL = WDTPW + WTHOLD;
26     flag42 = 1;
27   }
28 }
29
30 #pragma vector = PORT1_VECTOR
31 __interrupt void PORT1_a42(void){
32   if(P1IN & 0x01){
33     dosmth();
34   }
35   P1IFG &= ~(0xFF);
36 }
37
38 void dosmth(){
39   if(!flag42){
40     P1IE &= ~(0x01);
41     LED2OFF;
42     WDTCTL = WDTPW + WDTCNTCL + WDTSSEL + WDTTMSEL;
43     wait(50000);
44     P1IE = |= 0x01;
45   }
46 }
47
48 void aufgabe42(){
49   if(flag42){
50     ampel();
51   }else{
52     if(P4IN & 0x01){
53       P4IN &= ~(0x01);
54     } else {
55       P4IN |= 0x01;
```

```
56      }
57    }
58  }
59
60  void ampel(){
61    //Vgl Aufgabe 1
62  }
```

Erklärung:

In der Init aktivieren wir wie gewöhnlich den Taster zum lesen und die LED zum schreiben. Nun aktivieren wir noch die Interrupts und legen das InterruptEnabled Bit für den Taster und für den Watchdog an. Den Watchdog starten wir mit WDTTMSEL, wodurch bei auslösen nicht der Reset ausgeführt wird, sondern die ISR angesprungen wird.

Die ISR für den Watchdog scannt, ob die LED2 an ist. Ist sie an, reseten wir den Watchdog. Ist sie aus, schalten wir den Watchdog aus und ändern den Programmzustand (flag42), so dass danach die ampel ausgeführt wird.

Drücken wir den Taster, reseten wir den Watchdog, schalten die LED2 an.

In der Main, führen wir nun immer eine Ampel aus, wenn der Watchdoginterrupt die *flag42* gesetzt hat. Ist dies nicht der Fall, so Toggeln wir die LED1 immer.
Die flag42 wird nie zurückgesetzt. Dies soll dazu führen, dass nachträgliches drücken des Tasters die Bremsung nicht aufhalten kann. Darüber hinaus, wird der Watchdog auf hold gesetzt und sollte nicht mehr anspringen.

Beobachtung:

Wir beobachten, dass die LED1 erste getoggelt wird. Wenn wir zu diesem Zeit der Taster gedrückt wird, leuchtet für geschätzte 4s die LED2. Drücken wir den Taster nachdem die Ampel gestartet wurde, so passiert nichts mehr.