Mikroprozessorpraktikum WS 2011/12
Aufgabenkomplex: 5

Teilnehmer:

Marco Träger, Matr. 4130515
Alexander Steen, Matr. 4357549

Gruppe: Freitag, Arbeitsplatz: HWP 1

A 5.1 LPM und Interrupt

A 5.1.1 Starten Sie den Controller und überprüfen Sie messtechnisch den Stromverbrauch und die MCLK-Taktfrequenz.

Ohne weitere Programme einzufügen, d.h. mit leerer `while(1)`-Schleife, wird nun der Stromverbrauch und die Taktfrequenz gemessen. Um letzteres zu messen, legen wir das Taktsignal mittels folgendem Code an den Pin `P5.4`:

```
1  init511() {
2    P5SEL |= (1 << 4); //MCLK-Takt anlegen
3    P5DIR |= (1 << 4); //Als Ausgang nutzen
4  }
```

Diese Funktion wird vor der `while(1)`-Schleife aufgerufen und damit nur einmal ausgeführt (also wird damit die Messung nicht verfälscht). Eine genauere Erklärung des Codesegments findet sich in Protokoll 2, Sektion A 2.1. Der Stromverbrauch wird wie üblich mit dem Multimeter extern gemessen.
Die Messungen ergeben für das oben erklärte Szenario einen Stromverbrauch von 5.83 mA bei einer Taktfrequenz von ca. $7.36 \cdot 10^6$ Hz = 7.36 MHz.

Nun wird in die `while(1)`-Schleife der main-Funktion der Befehl `LPM4` eingefügt. Dieser Befehl bewirkt, dass der Mikrocontroller in den `LPM4`-Mode versetzt wird. Der Code sieht nun also folgendermaßen aus:

```
1  while(1) {
2    LPM4
3  }
```

Nach dem Start des Programmes werden nun, wie oben, Taktfrequenz und Stromverbrauch gemessen. Auffällig ist, dass die Messung eine Taktfrequenz von 0 Hz ergibt, also im `LPM4`-Mode scheinbar kein Taktzyklus mehr ausgeführt wird. Der Stromverbrauch ist dabei um 5.4 mA auf 0.43 mA gesunken.

Als nächstes fügen wir auf Port 1 eine ISR ein, die bei einem Tastendruck auf `P1.0` eine zehn Sekunden dauernde Warteschleife ausführt. Die `main`-Funktion bleibt dabei unverändert. Der Code, der diese Anforderungen realisiert, sieht folgendermaßen aus:

```
1  #pragma vector = PORT1_VECTOR
2  __interrupt void PORT1 (void) {
3    CLEAR(P1IFG, 0xFF);
4
5    if((P1IN & 0x01)) { //Taster gedrueckt
6      int i = 0;
7      while(i < 20) { // 20*0.5 Sekunden = 10 Sekunden
8        wait(50000); // Warte 0.5 Sekunden
9        ++i;
10     }
11   }
12 }
13
14 init511() {
15   // Initialisierung zur Taktmessung
16   P5SEL |= (1 << 4);
17   P5DIR |= (1 << 4);
18   // Taster-Initialisierung
19   P1DIR &= ~(0x01);
20   P1SEL &= ~(0x01);
21   P1IE |= (0x01); //Interrupts anschalten
22   P1IES &= ~(0x01);
23 }
```

Die ISR führt eine einfache Schleife aus, in der pro Durchlauf 0.5 Sekunden gewartet wird. Da die Schleife 20 mal durchläuft, erhalten wir eine insgesamte Wartezeit von zehn Sekunden. Der Parameter von `wait` wurde nicht direkt als zehn Sekunden gewählt, da wir sonst die zulässigen Bereich des Parameters überschreiten. Die Funktion `init511` setzt alle benötigten Register.
Die Messung ergibt nach dem Start des Programmes eine Taktfrequenz von 0 Hz bei einem Stromverbrauch von ca. 0.44 mA. Dies deckt sich mit unseren vorigen Messungen. Bei einem Tastendruck erhöhen sich die Messergebnisse für einen Zeitraum von knapp zehn Sekunden auf eine Taktfrequenz von 7.38 MHz bei einem Stromverbrauch von 4.08 mA.

A 5.2 Auto Shutdown mit einer ON/OFF Logik

A 5.2.1 Für die Umsetzung der in der Aufgabe beschriebenen Verhaltensweise, werden (1) Watchdog-Timer samt zugehöriger ISR und (2) Taster-ISR genutzt.

Die `main`-Funktion ruft im Folgenden die Funktion `aufgabe512` in einer Endlosschleife auf. Diese ist wie in der Aufgabe gefordert implementiert.

```
1  // Die Variable status bezeichnet den Modus, in dem wir
2  // uns befinden. Dabei ist status = 1, falls wir im LPM4-Modus
3  // sind. Sonst ungleich 1.
4  int status = 1;
5  // Zaehlen der Ticks
6  int tick = 0;
7
8  void init512() {
9    // Initialisieren aller noetigen Register,
10   // alle LEDs aus.
11   LED1ON;LED2ON;LED3ON;LEDON;
12   P5SEL |= (1 << 4);
13   P5DIR |= (1 << 4);
14   // Taster
15   P1DIR &= ~(0x01);
16     P1SEL &= ~(0x01);
17     P1IE |= (0x01);
18     P1IES &= ~(0x01);
19   // LEDs
20     P4DIR |= (0x07);
21     P4SEL &= ~(0x07);
22     P4OUT |= 0x07;
23 }
24
25 void aufgabe512() {
26   if (status == 1) { // LPM4-Modus
27     LED2OFF; LED3OFF;
28     LPM4;
29   } else { // aktiver Modus
30     LED2ON;
31   }
32   // Solange die Taste abfragen, wie sie
33   // gedrueckt wird.
34   while ((P1IN & 0x01)) {
35     if (tick > 2) {
36       // LPM4-Modus anfordern,
37       // Schleife verlassen
38       status = 1;
39       break;
40     }
41   }
42 }
```

Die Funktion `init512` wird dabei, wie immer, vor dem Aufruf von `aufgabe512` ausgeführt. Die Funktion `aufgabe512` tut exakt das, was in der Aufgabe gefordert ist: Falls wir im `LPM4`-Modus sein sollen (gdw. `status == 1`), werden die LEDs

ausgeschaltet und der Mikrocontroller in den LPM4-Modus versetzt. Ist dies nicht der Fall, so wird die LED P4.1 eingeschaltet. Danach wird der Taster abgefragt und überprüft, ob dieser länger als zwei Ticks gedrückt wurde. Ist dies der Fall, so wird dies als Abschaltvorgang interpretiert und die status-Variable auf eins gesetzt um in nächsten Schleifenzyklus (der main-Funktion) den LPM4-Modus zu aktivieren.
Die tick-Variable wird dabei sowohl von der ISR des Tasters als auch der ISR vom Watchdog-Timer beeinflusst:

```
1  #pragma vector = PORT1_VECTOR
2  __interrupt void PORT1 (void) {
3    P1IFG &= ~0xFF;
4
5    if((P1IN & 0x01)) { //Taste gedrueckt
6      if (status == 1) { // Waren/Sind wir gerade in LPM4-Mode?
7        status = 0; //Mode auf aktiv setzen: aufwachen
8
9        //Watchdog-Timer aktivieren, Ein-Sekunden-Timeout
10       WDTCTL = (WDTPW + WDTSSEL + WDTCNTCL);
11       // Interrupt auf Watchdog-Zimer-Intervall schalten
12       IE1 = WDTIE;
13     } else { //sind im aktiven Mode
14       // Es wurde etwas gemacht: tick zuruecksetzen
15       tick = 0;
16     }
17   }
18 }
19
20 #pragma vector = WDT_VECTOR
21 __interrupt void watchdog (void) {
22   IFG1 &= ~WDTIFG;
23   // Sekunde ist vergangen: tick inkrementieren
24   tick++;
25   // Blink-LED toggeln
26   P4OUT ^= 0x04;
27
28   // Sind wir im aktivem Modus und seit mehr als
29   // 60 Sekunden wurde die Taste nicht gedrueckt?
30   if ((status == 0) && (tick > 60)) {
31     // Dann LPM4-Mode anfordern
32     status = 1;
33     LED3OFF;
34   }
35 }
```

Die ISR des Watchdog-Timers hat zwei Aufgaben: Sie simuliert eine Uhr, in dem sie jede Sekunde die tick-Variable erhöht. Außerdem wird überprüft, ob seit 60 Sekunden (60 ticks) keine Aktion (Tastendruck) ausgeführt wurde. Ist dies der Fall so wird angenommen, dass das Modul gerade nicht aktiv benutzt wird und der Wechsel in den LPM4-Mode eingeleitet.
Die ISR des Tasters verhält sich, je nach Modus des Moduls, unterschiedlich: Wurde der Taster während des aktiven Modus gedrückt, so wird tick auf Null zurückgesetzt (um Aktivität kenntlich zu machen). Wird der Taster gedrückt, während man in LPM4-Modus ist, so wird das Modul aufgeweckt, status entsprechend gesetzt und der Watchdog-Timer eingeschaltet.
Damit ist ein Mechanismus zum automatischen ausschalten nach Inaktivität bzw. zum Ein- und Ausschalten via Tastendruck implementiert.

Der Test des Programmes zeigt korrektes Verhalten und bestätigt damit die Erklärungen. Auch Messungen der Taktfrequenz und damit Beobachtungen über das Ein- und Austretens aus dem LPM4-Mode, belegen die Funktion des Programmes.

A 5.3 Wake Up binär gesteuert

A 5.3.1 Unter Benutzung des Tasters als Sensor, soll ein Wake-Up-Mechanismus umgesetzt werden, der auf Zustandsänderungen des Sensors entsprechend reagiert.

Das nachfolgende Programm setzt das gewünschte Verhalten um. Die Initialisierung der Register und das Erlauben von Interrupt passiert in der Funktion `init513`, die wie immer vor der Funktion `aufgabe513` aufgerufen wird. Die `main`-Funktion enthält dann eine Endlos-Schleife, in der `aufgabe513` aufgerufen wird.

```
1  // Diese Variable codiert den aktuellen Status:
2  // status == 1 bedeutet, dass wir in den aktiven Modus wechseln
3  // status == 0 bedeutet, dass wir in den LPM4-Modus wechseln
4  unsigned char status = 1;
5
6  void init513() {
7    // Alle LEDs aus, Register initialisieren
8    LED1OFF;LED2OFF;LED3OFF;LED0FF;
9    P5SEL |= (1 << 4);
10   P5DIR |= (1 << 4);
11   //Taster interruptfaehig
12   P1DIR &= ~(0x01);
13     P1SEL &= ~(0x01);
14     P1IE |= (0x01);
15   // LED-Register einrichten
16     P4DIR |= (0x07);
17     P4SEL &= ~(0x07);
18     P4OUT |= 0x07;
19   // Interrupt-Flanke auf low-hi
20     P1IES &= ~(0x01); //LH
21 }
22
23 #pragma vector = PORT1_VECTOR
24 __interrupt void PORT1 (void) {
25   P1IFG &= ~0xFF;
26
27   if ((P1IES & 0x01) == 1) {//HL ist Interruptquelle
28       status = 1; //aktiver Mode
29       P1IES &= ~(0x01);  // LH ist Interruptquelle
30       LPM4_EXIT; // Aus LPM4-Mode austreten
31     } else {
32       status = 0; //LPM anfordern
33       P1IES |= (0x01); //HL ist nun Interruptquelle
34     }
35 }
36
37 void aufgabe513() {
38   if (status == 0) { //LPM angefordert
39     LED1OFF;LED2OFF;LED3OFF;
40     LPM4;
41   } else {  //aktiver Mode
42     // Ampelsequenz
43     P4OUT = ROT;
44     wait(waittime);
45     P4OUT = ROTGELB;
46     wait(waittime);
47     P4OUT = GRUEN;
48     wait(waittime);
49     P4OUT = GELB;
50     wait(waittime);
51     P4OUT = ROT;
52     wait(waittime);
53   }
54 }
```

Die ISR des Tasters überprüft, ob der Interrupt durch eine LH- oder HL-Flanke ausgelöst wurde und entscheidet entsprechend, ob in den aktiven Modus oder in den `LPM4`-Modus gewechselt werden soll. Die Funktion `aufgabe513` setzt dann entsprechend der Belegung von `status` entweder den `LPM4`-Modus um oder schaltet eine Ampelsequenz. Die Variablen `waittime` und die Makros der Ampelsequenz

wurden dabei aus Protokoll 1 übernommen.
Das Programm wurde erfolgreich getestet, das Wechseln in den Low-Power-Mode wurde durch Taktfrequenzmessungen bestätigt, ebenso liefen die Ampelsequenzen erfolgreich durch.