Mikroprozessorpraktikum WS 12/13

Carlos Martín Nieto, Simon Hohberg, Tu Tran

February 7, 2013

5 LPM Mode

5.1 LPM und Interrupt

5.1.1 Starten Sie den Controller und überprüfen Sie messtechnisch den Stromverbrauch und die MCLK-Taktfrequenz (wie in A 2.1.1 und A2.2.1).
Fügen Sie in der while(1) Schleife der main() Funktion einen Befehl ein, der den Mikrocontroller in den LPM4 Mode setzt.
Starten Sie das Programm. Was bewirkt der Befehl? Wie verhalten sich die Taktfrequenz und der Stromverbrauch?

- Normal:

Stromverbrauch	Taktfrequenz
6,25 mA	7,35 MHz

- LPM4 Mode:

Stromverbrauch	Taktfrequenz
0,97 mA	-

Keine Takfrequenz, da alle clocks und die CPU abgeschaltet werden.

Programmieren Sie den Port1 in der Form, dass bei einem Druck auf die Taste (P1.0) ein Interrupt ausgelöst wird. Die while(1) Schleife bleibt unverändert, es befindet sich nur die LPM4 Anweisung in der Schleife. Realisieren Sie in der ISR des PORT1 eine 10 Sekunden dauernde Warteschleife.

```
1  #define SLEEP_QUANTUM 10000
2  /*
3   * schlaeft fuer n Sekunden
4   */
5  #define SLEEP(n) do { \
6       long time = n * 100000; /* wait() sleeps 10*n microseconds */ \
7       while(time > SLEEP_QUANTUM) { \
8            wait(SLEEP_QUANTUM); \
9            time -= SLEEP_QUANTUM; \
10      } \
11      wait(time); \
12 } while(0)
13
14 // P1.0 Input
15 BIT_CLR(P1DIR, TASTE_LINKS);
```

```
16
17 BIT_SET(P1IES, TASTE_LINKS); // HL an linker taste loest interrupt aus
18 BIT_SET(P1IE, TASTE_LINKS);  // interrupts fuer linke taste erlauben
19 BIT_CLR(P1IFG, TASTE_LINKS); // interrupt flag loeschen
20
21 LED_OFF(LED_ALL);
22
23 // takt ausgeben, zum debuggen
24 BIT_SET(P5SEL, 1 << 4);
25 BIT_CLR(P5DIR, 1 << 4);
26
27 _bis_SR_register(GIE);
28
29 //===Hier die Endlosschleife quasi das Betriebssystem==================
30   while(1){
31
32         LPM4;
33
34     }   // Ende der Endlosschleife
35 }       // Ende Main
36 //====Ende des Hauptprogramms ========================================
37
38 #pragma vector = PORT1_VECTOR
39 __interrupt void PORT1(void)
40 {
41     if (P1IFG & TASTE_LINKS) {
42
43         LPM4_EXIT;
44
45         SLEEP(10);
46
47         BIT_CLR(P1IFG, TASTE_LINKS);
48     }
49 }
```

Dokumentieren Sie die Beobachtungen (Stromverbrauch und Taktfrequenz) zum Zeitpunkt des Tastendruckes.

Stromverbrauch	Taktfrequenz
5,16 mA	7,36 MHz

Erläutern Sie die Abläufe nach dem Starten des Programms im Detail und die Zustände des Mikrocontrollers.

Beim Starten des Programms befindet sich der Mikrocontroller im normalen Zustand. Zunächst wird die durch das Programm spezifizierte Initialisierung durchgeführt und danach wird schließlich die Endlosschleife ausgeführt. Hier wird dann der LPM4-Befehl ausgeführt, sodass der Mikrocontroller diesen Modus einnimmt. Dies beinhaltet das Abschalten aller Clocks, was dazu führt, dass das Programm "pausiert" wird. Der Mikrocontroller befindet sich nun sozusagen im Schlafmodus.

Wird nun der linke Taster gedrückt, löst dies den dazugehörigen Interrupt aus. In der Interruptserviceroutine wird der Mikrocontroller dann zunächst mit dem Befehl LPM4_EXIT "aufgeweckt", also alle Clocks wieder aktiviert.

Nachdem die Interruptserviceroutine abgearbeitet wurde, wird die Ausführung der while-Schleife wieder aufgenommen. Schließlich wird wieder der LPM4-Befehl ausgeführt, so dass der Mikrocontroller wieder "schlafen

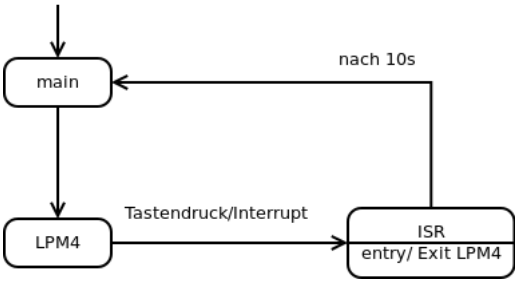

Figure 1: Programmzustand

gelegt" wird bis das nächste Mal der Taster gedrückt wird.

5.2 Auto Shutdown

5.2.1 Für die Umsetzung der oben beschriebenen Verhaltensweise, werden der Watchdog in einer Timer Anwendung und parallel dazu zwei Interruptquellen genutzt. Der Taster (P1.0) soll als Bedienelement genutzt werden.

```
1  ....
2  main(void);                    //Haupprogramm
3  void print_value(void);
4
5  // status
6  enum {
7       MODE_LPM4,
8       MODE_ACTIVE
9  } status;
10
11 int tick = 0;
12
13 #define BIT_SET(a,b) ((a) |= (b))
14 #define BIT_CLR(a,b) ((a) &= ~(b))
15 #define BIT_TOGGLE(a,b) ((a) ^= (b))
16
17 //====Hauptprogramm=============================================
18
19
20 main(void)
21      {
22 //===Hier sollten Variablen deklariert werden ==================
23      //unsigned char i = 0;
24      //char text[60];
25      //int x,y;
26
27 //===Hier die notwendigen Initialisierungsschritte ============
28 //=(1)== Port-Initialisierung ================================
29      init_Port();                  // Initialisierung der Port Register
30
31 //(2)=== Clock-System-Initialisierung ========================
32      //== XT2() oder Dco() als Taktquelle einstellen
33      //== durch Ein- oder Auskommentieren
34      //== DCO ist bei LPM Einsatz bevorzugt muß zyklisch kalibriert werden
35      //== XT2 ist quarzstabil muß nicht zyklisch kalibriert werden
```

```
36      //
37      //XT2 ();                       // XT2 Taktquelle aktivieren mit 7.3728MHz
38      DCO ();                     // Dco Taktquelle aktivieren mit 7.3728MHz
39      //                              beachte DELTA
40
41  //=(3)== Timer-Initialisierung= ======================================
42      init_Timer_A();             // Init Timer für Sekundeninterrupt
43                                  // !! noch leere Funktion
44
45  //=(4)== USART-Initialisierung ======================================
46      init_UART1();               // UART-RS232 mit 115.2kBit/s initialisieren
47                                  // !! noch leere Funktion
48
49  //=(5)== CC1100-Transceiver-Initialisierung ==========================
50      init_UART0_SPI();           // CC1100 SPI UART initalisieren
51      init_CC1100_POWERDOWN();              // CC1100 init und in RX Mode setzen
52                                  // !!!Interrupte sind ab jetzt freigegeben!!
53      //== Adresse und Funkkanal des Transceivers setzen
54      //== für die Arbeitsplaetze HWPx (x=1...10) sollten
55      //== ID=x und channnel=x gesetzt werden
56      ID = 1;                             // Adresse
57      setUid(ID);                         // Adresse im Transceiver setzen
58      channel = 1;                        // Funkkanal
59      switchFreq(channel);                // Funkkanal im Transceiver setzen
60      //== Soll der Transceiver genutzt werden müssen folgende zwei Zeilen
61      //== auskommentiert werden:
62      init_CC1100_IDLE();         // CC1100 in den IDLE Mode setzen
63      init_CC1100_POWERDOWN();// CC1100 in den PowerDown Mode setzen
64
65  //=(6)== LCD-Display-Initialisierung ================================
66      dogm_reset();           // Hardware Reset des LCD Controllers
67      dogm_init();            // Initialisierung der LCD Controller Register
68      lcd_clear(WHITE);       // Grafikspeicher auf dem MSP430 löschen
69      //lcd_string(BLACK, 15, 25, "MSP430-GESTARTET!");   // Textausgabe
70      lcd_paint();            // Grafikspeicher auf das LCD Display ausgeben
71
72
73  #define LED_ROT (0x01)    // 0 0 1 P4.0
74  #define LED_GELB (0x02)   // 0 1 0 P4.1
75  #define LED_GRUEN (0x04) // 1 0 0 P4.2
76  #define LED_ALL (LED_ROT | LED_GELB | LED_GRUEN)
77
78  #define LED_ON(led) (BIT_CLR(P4OUT, led))
79  #define LED_OFF(led) (BIT_SET(P4OUT, led))
80  #define LED_TOGGLE(led) (BIT_TOGGLE(P4OUT, led))
81
82  #define IS_LED_ON(led) (!(P4OUT & led))
83
84  #define TASTE_LINKS (0x1)
85  #define TASTE_RECHTS (0x2)
86
87  #define SLEEP_QUANTUM 10000
88  #define SLEEP(n) do {         /* sleep for n seconds */ \
89      long time = n * 100000; /* wait() sleeps 10*n microseconds */ \
90      while(time > SLEEP_QUANTUM) { \
91          wait(SLEEP_QUANTUM); \
92          time -= SLEEP_QUANTUM; \
93      } \
94      wait(time); \
95  } while(0)
96
97  // P1.0 Input
```

```
98  BIT_CLR(P1DIR, TASTE_LINKS);
99
100 BIT_CLR(P1IES, TASTE_LINKS); // LH an linker taste loest interrupt aus
101 BIT_SET(P1IE, TASTE_LINKS);  // interrupts fuer linke taste erlauben
102 BIT_CLR(P1IFG, TASTE_LINKS); // interrupt flag loeschen
103
104 LED_OFF(LED_ALL);
105
106 status = MODE_LPM4;
107 tick = 0;
108
109 _bis_SR_register(GIE);
110
111 //===Hier die Endlosschleife quasi das Betriebssystem==================
112   while(1){
113
114         // LPM mode
115         if (MODE_LPM4 == status) {
116             LED_OFF(LED_ALL);
117             // disable watchdog interrupt
118             BIT_CLR(IE1, WDTIE);
119             // stoppe watchdog
120             WDTCTL = WDTPW + WDTHOLD;
121             LPM4;
122         }
123
124         // aktiver mode
125         if (MODE_ACTIVE == status) {
126             LED_ON(LED_GELB);
127         }
128
129         // busy waiting bis taster losgelassen wird
130         while (P1IN & TASTE_LINKS) {
131             if (tick > 2) {
132                 status = MODE_LPM4;
133                 break;
134             }
135         }
136
137     }   // Ende der Endlosschleife
138 }       // Ende Main
139 //====Ende des Hauptprogramms ========================================
140
141 #pragma vector = PORT1_VECTOR
142 __interrupt void PORT1(void)
143 {
144     if (P1IFG & TASTE_LINKS) {
145
146         tick = 0;
147
148         if (MODE_LPM4 == status) {
149             // zu aktivem modus wechseln
150             status = MODE_ACTIVE;
151             LPM4_EXIT;
152             // watchdog programmieren
153             // passwort, clear counter, clock source: ACLK , interval timer mode
154             // implizit: timer interval 32768 -> 1s
155             WDTCTL = WDTPW + WDTCNTCL + WDTSSEL + WDTTMSEL;
156             // enable interrupt
157             BIT_SET(IE1, WDTIE);
158         }
159
```

```
160             BIT_CLR(P1IFG, TASTE_LINKS);
161         }
162 }
163
164 #pragma vector = WDT_VECTOR
165 __interrupt void WDT(void)
166 {
167         tick++;
168         LED_TOGGLE(LED_GRUEN);
169         // nach einer minute automatisch in schlafmodus wechseln
170         if (MODE_ACTIVE == status && tick > 60) {
171             status = MODE_LPM4;
172             LED_OFF(LED_GRUEN);
173         }
174 }
175 ...
```

5.3 Wake Up

5.3.1 Der Taster an der Portleitung (P1.0) ersetzt uns den Sensor. Per ISR soll ein Zustandswechsel an der Portleitung erfasst werden. Eine LH-Flanke setzt eine Statusvariable auf 1 und eine HL-Flanke setzt die Statusvariable auf 0.

```
1  // P1.0 Input
2  BIT_CLR(P1DIR, TASTE_LINKS);
3
4  BIT_SET(P1IES, TASTE_LINKS); // HL an linker taste loest interrupt aus
5  BIT_SET(P1IE, TASTE_LINKS);  // interrupts fuer linke taste erlauben
6  BIT_CLR(P1IFG, TASTE_LINKS); // interrupt flag loeschen
7
8  LED_OFF(LED_ALL);
9
10 // takt ausgeben, zum debuggen
11 BIT_SET(P5SEL, 1 << 4);
12 BIT_CLR(P5DIR, 1 << 4);
13
14 _bis_SR_register(GIE);
15
16 //===Hier die Endlosschleife quasi das Betriebssystem==================
17   while(1){
18
19         // LPM mode
20         if (0 == status) {
21             LED_OFF(LED_ALL);
22             LPM4;
23         }
24
25         // aktiver mode
26         if (1 == status) {
27
28             // ampelsequenz
29             LED_ON(LED_ROT);
30             SLEEP(2);
31             LED_ON(LED_GELB);
32             SLEEP(1);
33             LED_OFF(LED_ROT);
34             LED_OFF(LED_GELB);
35             LED_ON(LED_GRUEN);
36             SLEEP(2);
```

```
37              LED_OFF(LED_GRUEN);
38              LED_ON(LED_GELB);
39              SLEEP(1);
40              LED_OFF(LED_GELB);
41          }
42
43      }   // Ende der Endlosschleife
44  }       // Ende Main
45  //====Ende des Hauptprogramms ====================================
46
47  #pragma vector = PORT1_VECTOR
48  __interrupt void PORT1(void)
49  {
50      if (P1IFG & TASTE_LINKS) {
51
52          if (P1IES & TASTE_LINKS) {
53              // high - low flanke aktiviert
54              status = 1; // aktiver mode
55              LPM4_EXIT;
56              // setze interrupt fuer low - high
57              BIT_CLR(P1IES, TASTE_LINKS);
58          } else {
59              // low - high flanke aktiviert
60              status = 0; // low power mode
61              // setze interrupt fuer high - low
62              BIT_SET(P1IES, TASTE_LINKS);
63          }
64
65          BIT_CLR(P1IFG, TASTE_LINKS);
66      }
67  }
```