Mikroprozessorpraktikum WS 12/13

Carlos Martín Nieto, Simon Hohberg, Tu Tran

February 14, 2013

8 Voltmeter

8.1 Der Mikrocontroller als Voltmeter

8.1.1 Entwickeln Sie ein Programm das folgende Funktionalität besitzt:

• Timergesteuert und im Sekundentakt soll eine A/D-Wandlung statt finden
• Der gewandelte Wert soll in dem Format "U=x,xx V" auf dem LCD-Display ausgegeben werden.
• Bei U<1V gelbe LED (P4.1) an
• Bei 1V < U < 2V grüne LED (P4.2) an
• Bei U > 2V rote LED (P4.0) an

```
1  // waehle P6.7 als input
2  P6SEL |= BIT7;
3  // schalte wandler an, waehle "multiple sample and conversion", SHT
       = 0 -> 4 cycles
4  ADC12CTL0 = ADC12ON + MSC;
5  // ADC12 clock, repeat single channel
6  ADC12CTL1 = CONSEQ1 + SHP;
7  // waehle channel 7
8  ADC12MCTL0 = INCH_7;
9  // loesche interrupt flag
10 BIT_CLR(ADC12IFG, 1);
11 // erlaube interrupt
12 BIT_SET(ADC12IE, 1);
13
14 // erlaube conversion
15 ADC12CTL0 |= ENC;
16 // starte conversion
17 ADC12CTL0 |= ADC12SC;
18
19 _bis_SR_register(GIE);
20 //===Hier die Endlosschleife quasi das Betriebssystem
       ==================
21     while(1){
22     }    // Ende der Endlosschleife
23 }        // Ende Main
```

```
24 //====Ende des Hauptprogramms
       ======================================
25
26 void print_buf(const char *str)
27 {
28     do {
29         U1TXBUF = *str;
30         /*
31          * (115200 / 8)**-1 = 70us pro Zeichen
32          */
33         wait(7);
34     } while(*str++);
35 }
36
37 static float volts;
38
39 #pragma vector = ADC12_VECTOR
40 __interrupt void ADC12(void)
41 {
42     char buffer[16];
43     if (ADC12IFG & 1)
44     {
45         volts = ADC12MEM0/4096.0f;
46         volts *= 3.3f;
47
48         LED_OFF(LED_ALL);
49         if (volts < 1)
50         {
51             LED_ON(LED_GELB);
52         }
53         else if (volts < 2)
54         {
55             LED_ON(LED_GRUEN);
56         }
57         else
58         {
59             LED_ON(LED_ROT);
60         }
61
62         sprintf(buffer, "U=%.2f V", volts);
63         lcd_clear(WHITE);
64         lcd_string(BLACK, 15, 25, buffer);
65         lcd_paint();
66         BIT_CLR(ADC12IFG, 1);
67     }
68 }
```

8.2 Sensor

8.2.1 In dieser Aufgabe sollen Sie Beschleunigungswerte als analoge Größen erfassen und Visualisieren. Entwickeln Sie ein Programm das folgende Funktionalität besitzt:

- Initialisierung der Portleitungen

- Einstellung des 1,5g Messbereiches des MMA7260A
- Timergesteuerte AD-Wandlung der 3D-Beschleunigungswerte (ax-, ay- und az-Werte) im 0,5 Sekundentakt
- Ausgabe der Werte als Zeichenkette auf die serielle Schnittstelle
- Zeichenkettenstruktur:
- <ax><HT><ay><HT><az><CR><LF>
- ax, ay und az Werte als ASCII Zeichenkette im Bereich von 0...4096
  - HT - Horizontal Tab
  - CR - Carriage Return
  - LF - Line Feed
  - Zeigen Sie für alle drei Achsen auf dem LCD-Display die Ergebnisse der A/D-Wandlung als Zahl im Bereich von 0...4095 an.

E

```
1  // waehle ACLK
2  BIT_SET(TBCTL, TBSSEL0);
3  BIT_CLR(TBCTL, TBSSEL1);
4
5  // waehle Set/Reset
6  //BIT_SET(TBCCTL1, OUTMOD1 + OUTMOD0);
7
8  // setze divider auf 8
9  BIT_SET(TBCTL, ID0 | ID1);
10
11 // wahle count up mode
12 BIT_SET(TBCTL, MC0);
13 BIT_CLR(TBCTL, MC1);
14
15 // setze taktanzahl fuer eine halbe sekunde
16 TBCCR0 = 2048; // eine sekunde: takt=32000 / divider=8
17
18 // loesche interrupt flag fuer timer
19 BIT_CLR(TBCCTL0, CCIFG);
20
21 TBR = 1;
22
23 // setze Pin 5.4, 5.5, 5.6 als output
24 BIT_SET(P5DIR, BIT4 + BIT5 + BIT6);
25 // Beschleunigungssensor g-Range 1.5g, Sensitivity 800mV/g
26 BIT_CLR(P5OUT, BIT4);
27 BIT_CLR(P5OUT, BIT5);
28 // sleep mode off
29 BIT_SET(P5OUT, BIT6);
30
31 // waehle P6.0, P6.1, P6.2 als input
32 P6SEL = BIT0 + BIT1 + BIT2;
33 // schalte wandler an, waehle "multiple sample and conversion", SHT
       = 0 -> 4 cycles
```

```
34  ADC12CTL0 = ADC12ON + MSC;
35  // ADC12 clock, sequence of channels
36  ADC12CTL1 = CONSEQ0 + SHP;
37  // waehle channel A0, A1, A2
38  ADC12MCTL0 = INCH_0;
39  ADC12MCTL1 = INCH_1;
40  ADC12MCTL2 = INCH_2 + EOS;
41  // loesche interrupt flag
42  BIT_CLR(ADC12IFG, BIT2);
43  // erlaube interrupt
44  BIT_SET(ADC12IE, BIT2);
45  // erlaube timer interrupt
46  BIT_SET(TBCCTL0, CCIE);
47
48  // erlaube conversion
49  ADC12CTL0 |= ENC;
50
51  _bis_SR_register(GIE);
52  //===Hier die Endlosschleife quasi das Betriebssystem
        ==================
53      while(1){
54      }    // Ende der Endlosschleife
55  }        // Ende Main
56  //====Ende des Hauptprogramms
        ========================================
57
58  void print_buf(const char *str)
59  {
60      do {
61          U1TXBUF = *str;
62          /*
63           * (115200 / 8)**-1 = 70us pro Zeichen
64           */
65          wait(7);
66      } while(*str++);
67  }
68
69  #pragma vector = TIMERB0_VECTOR
70  __interrupt void TIMERB0(void)
71  {
72      // starte conversion
73      ADC12CTL0 |= ADC12SC;
74  }
75
76  #pragma vector = ADC12_VECTOR
77  __interrupt void ADC12(void)
78  {
79      char buffer[32];
80      int x, y, z;
81      if (ADC12IFG & BIT2)
82      {
83          x = ADC12MEM0;
84          y = ADC12MEM1;
85          z = ADC12MEM2;
86
87          lcd_clear(WHITE);
88          sprintf(buffer, "ax=%d", x);
```

```
89              lcd_string(BLACK, 16, 16, buffer);
90              sprintf(buffer, "ay=%d", y);
91              lcd_string(BLACK, 16, 25, buffer);
92              sprintf(buffer, "az=%d", z);
93              lcd_string(BLACK, 16, 33, buffer);
94
95              sprintf(buffer, "%d\t%d\t%d\r\n", x, y, z);
96
97              print_buf(buffer);
98
99              lcd_paint();
100             BIT_CLR(ADC12IFG, BIT2 + BIT1 + BIT0);
101         }
102 }
```

So werden die Beschleunigungswerte berechnet:
Neuer Wert $= (\frac{Beschleunigungswerte}{4095} \cdot 3 - 1.5) \cdot 9.81 \frac{m}{s^2}$

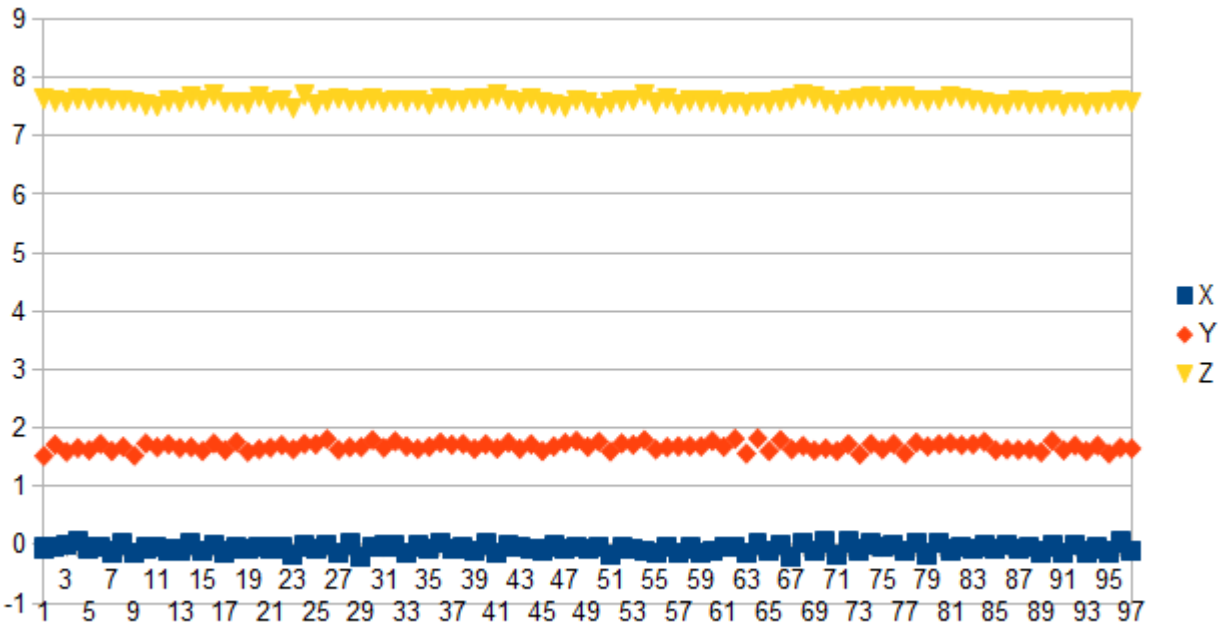

Figure 1: Beschleunigungswerte über der Excel-Tabelle

8.3 Touchscreen

8.3.1 Bei Berührung des Touchscreens soll eine ISR gestartet werden. In der ISR sollen die Koordinaten der Touchscreenberührung auf das LCD-Display ausgegeben werden. Solange der Touchscreen berührt wird soll die Anzeige ständig aktualisiert werden und die LED (P4.1) leuchten. Die obere linke Ecke soll die Koordinaten (0,0) besitzen. In den Achsen sind folgende Bereiche möglich:

- **X = 0..127**
- **Y = 0..63**

Initialisierung wie oben.

```
1  void ts_reset_pins(void)
2  {
3    // alle Leitungen auf Eingang
4      TS_TIP_DIR_IN;
5      TS_YP_DIR_IN; TS_YM_DIR_IN;
6      TS_XP_DIR_IN; TS_XM_DIR_IN;
7  }
8
9  // Einlesen Touch-Event
10 void ts_listen(void)
11 {
12     ts_reset_pins();
13     // das Ausgangsregister vorbereitend setzen
14     TS_TIP_1; // YP Y-Achse wird über einen PullUp Widerstand auf 1
           gezogen
15     TS_XM_0; // XM X-Achse wird auf 0 gesetzt
16     // Die Ausgaenge jetzt freigeben
17     TS_TIP_DIR_OUT; // YP auf 1
18     TS_XM_DIR_OUT;  // XM auf 0
19
20     // erlaube Interrupt in P1.6 bei wechsel auf 0
21     BIT_SET(P1IES, BIT_YM); // HL
22     BIT_CLR(P1IFG, BIT_YM);
23     BIT_SET(P1IE, BIT_YM);
24 }
25
26 // Bereite TS fuer Einlesen von X vor
27 void ts_cfg_x(void)
28 {
29     ts_reset_pins();
30     // die Ausgangsregister vorbereitend setzen
31     TS_XP_1; // XP X-Achse rechts auf 1
32     TS_XM_0; // XM X-Achse links auf 0
33     // Die Ausgaenge freigeben
34     TS_XP_DIR_OUT;  // XP auf 1 freigeben
35     TS_XM_DIR_OUT;  // XM auf 0 freigeben
36 }
37
38 void ts_cfg_y(void)
39 {
40     ts_reset_pins();
41     // die Ausgangsregister vorbereitend setzen
```

```
42      TS_YP_1; // YP Y-Achse oben auf 1
43      TS_YM_0; // YM Y-Achse unten auf 0
44      // Die Ausgaenge freigeben
45      TS_YP_DIR_OUT;   // YP auf 1 freigeben
46      TS_YM_DIR_OUT;   // YM auf 0 freigeben
47  }
48
49  int main(void)
50  {
51      [...]
52      ts_listen();
53
54
55      _bis_SR_register(GIE);
56  //===Hier die Endlosschleife quasi das Betriebssystem
        ====================
57      while(1){
58      }    // Ende der Endlosschleife
59  }        // Ende Main
60  //====Ende des Hauptprogramms
        ======================================
61
62
63  #pragma vector = PORT1_VECTOR
64  __interrupt void PORT1(void)
65  {
66      char buffer[32];
67      int x, y;
68      static const int xmin = 660;
69      static const int xwidth = 3475 - 660;
70      static const int ymin = 848;
71      static const int ywidth = 3343 - 848;
72
73      LED_ON(LED_GELB);
74
75      while (P1IFG & BIT6 || !(P1IN & BIT_YM)) {
76
77          // loesche touch interrupt flag und schalte interrupt aus
78          BIT_CLR(P1IFG, BIT6);
79          BIT_CLR(P1IE, BIT_YM);
80
81          // verbinde YP
82          BIT_SET(P6SEL, BIT_YP);
83          BIT_CLR(P6DIR, BIT_YP);
84
85          // lade konfiguration zum Lesen von x
86          ts_cfg_x();
87
88          // schalte Wandler an
89          ADC12CTL0 = ADC12ON;
90          // konfiguriere AD Wandler
91          ADC12CTL1 = 0;
92          ADC12MCTL0 = INCH_4;
93          // starte Wandlung
94          ADC12CTL0 |= ENC + ADC12SC;
95          // stoppe Wandlung
96          BIT_CLR(ADC12CTL0, ADC12SC);
```

```
97
98          // Warte, bis die Konversion abgeschlossen ist
99          while(ADC12CTL1 & ADC12BUSY);
100         x = ADC12MEM0;
101
102         // Normalisieren auf 0...(xwidth)
103         x -= xmin;
104         x = ((float)1 - (((float)(xwidth - x))/((float)(xwidth))))
                * 127;
105
106         BIT_CLR(P6SEL, BIT_YP);
107
108
109         // lade konfiguration zum Lesen von y
110         ts_cfg_y();
111
112         // verbinde XP
113         BIT_SET(P6SEL, BIT_XP);
114         BIT_CLR(P6DIR, BIT_XP);
115
116         // schalte Wandler an
117         ADC12CTL0 = ADC12ON;
118         // konfiguriere AD Wandler
119         ADC12CTL1 = 0;
120         ADC12MCTL0 = INCH_5;
121         // starte Wandlung
122         ADC12CTL0 |= ENC + ADC12SC;
123         // stoppe Wandlung
124         BIT_CLR(ADC12CTL0, ADC12SC);
125
126         // Warte, bis die Konversion abgeschlossen ist
127         while(ADC12CTL1 & ADC12BUSY);
128         y = ADC12MEM0;
129         // Normalisieren auf 0...(ywidth)
130         y -= ymin;
131         y = (((float)(ywidth - y))/((float)ywidth)) * 63;
132
133         // schalte Wandler aus
134         ADC12CTL0 = 0;
135
136         BIT_CLR(P6SEL, BIT_XP);
137
138         lcd_clear(WHITE);
139         sprintf(buffer, "x=%d y=%d", x, y);
140         lcd_string(BLACK, 8, 15, buffer);
141         lcd_paint();
142
143         ts_listen();
144         BIT_SET(P1IE, BIT_YM);
145         BIT_CLR(ADC12IFG, 1);
146     }
147
148     LED_OFF(LED_GELB);
149 }
```